SpaceX Falcon 9 First Launch Stage Success Prediction

by Justin Tapia

Table of Contents

Executive Summary

Problem Statement

- Predict the success of SpaceX's Falcon 9 first stage launch.

Methodology Summary:

- **Data Collection and Wrangling**: Utilizing the SpaceX API and web scraping from SpaceX's Wikipedia page.
- **Analysis Tools**: Visual/SQL EDA, SQLite database integration, Folium geospatial analysis, and Dash interactive data visualization.
  - Each EDA is designed to facilitate problem revisitation and other question provocation.
- **Predictive Modeling**: Comparison of Decision Trees, KNN, Logistic Regression, and SVM using GridSearchCV for optimization.
  - Eight models - Improved models built off initial model skeletons, where **logreg_cv2** and **tree_cv2** model metrics perform significantly higher than the others.

Introduction

Overview

- Space X's Falcon 9's partially reusable two-stage rocket is designed with a focus on reducing the cost of space travel through reusability. Advertised on its website with a cost of $67 million, its primary point of cost savings is its first stage reusability. Predicting the first stage landing outcomes, cost optimization strategies can be implemented to streamline the launch process and make space travel more affordable. The student is a representative of SpaceY tasked with creating a machine learning pipeline to predict first launch success

Objective

- Generate implicative insight to facilitate cost analysis, empowering SpaceY as an aerospace competitor to develop superior investment strategies and competition in relation to SpaceX's determinant factors of Falcon9 first stage launch success.

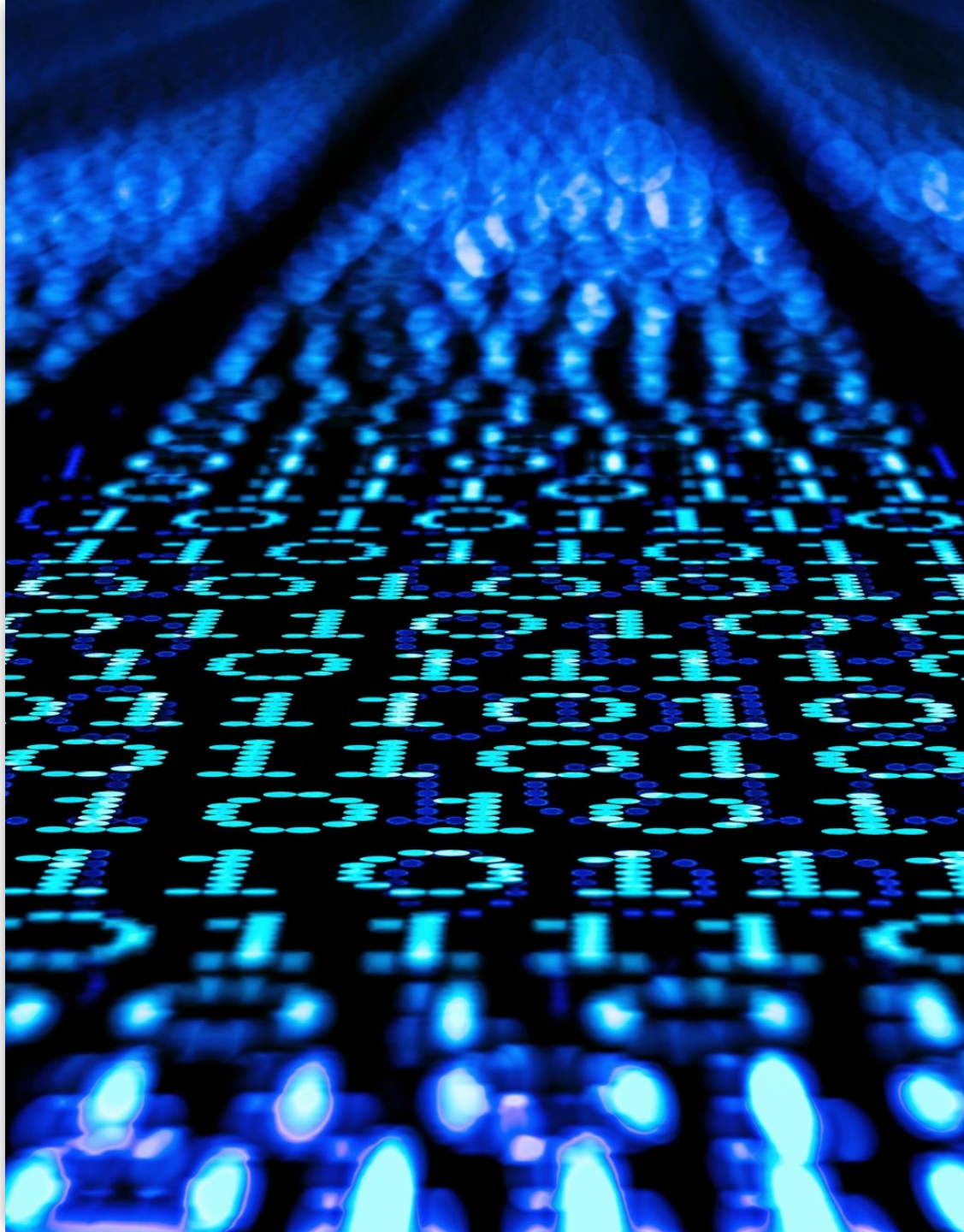
Methodology Overview

Methodology

SpaceX API Data Collection

Steps

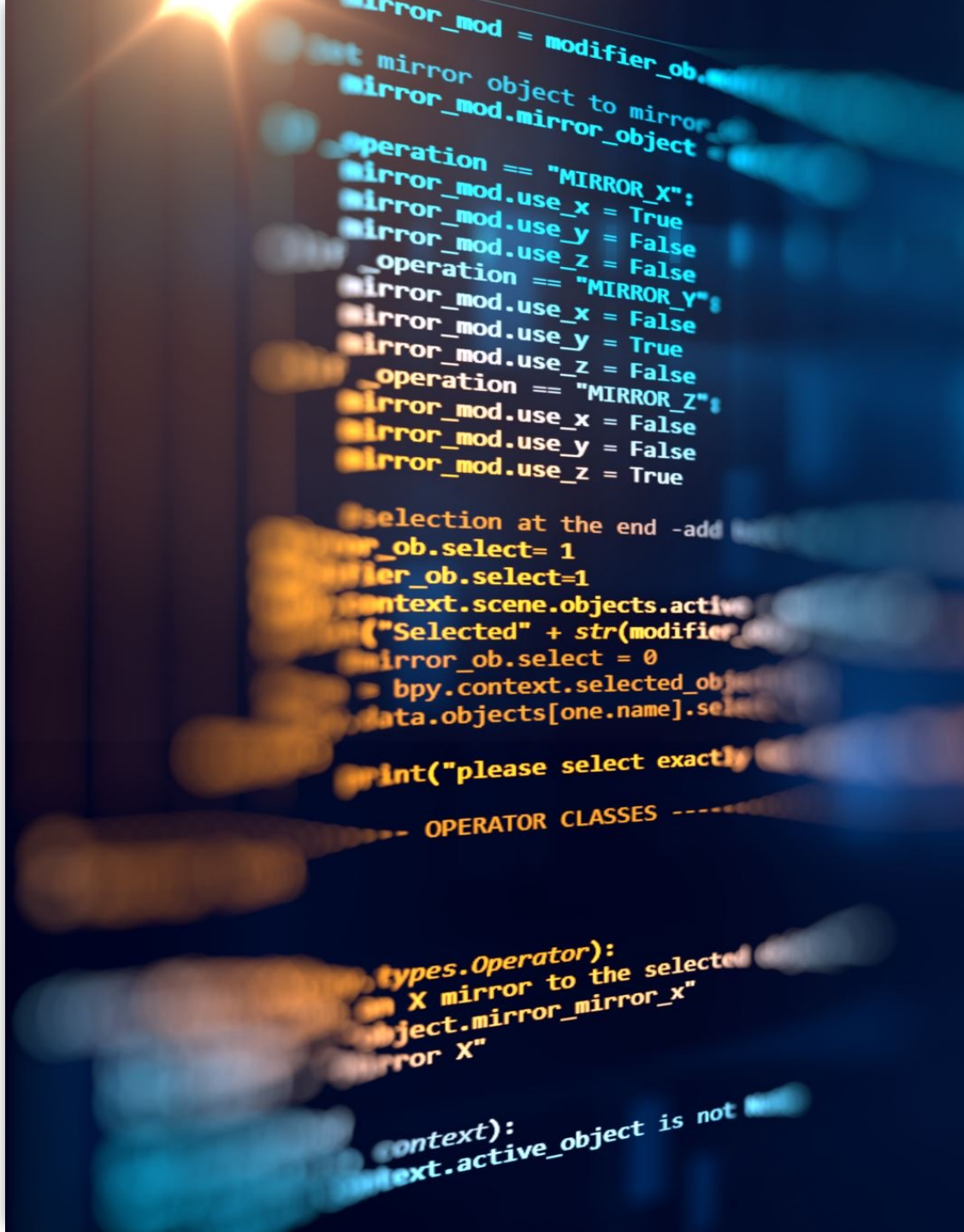
- Request and parse SpaceX launch data using GET request
- Create custom functions to extract specific variables from dataset
- Combine gathered columns into a dictionary
- Create dataframe from the dictionary
- Filter dataframe for Falcon9 launches
- Replace Payload Mass null values with their calculated mean
- Export data to **spacex_api.csv**



SpaceX Wiki Web Scrape

Steps

- Identify Wikipedia page as suitable candidate for data collection
- Request the HTML page as an HTTP response, set to BeautifulSoup object
- Extract column names from HTML table headers
- Create dictionary with keys from the extracted column names
- Parse HTML table and fill dictionary usingcustom function
- Set to dataframe
- Export data to **spacex_ws.csv**



Data Wrangling

Calculated:

- Launches per Site
- Occurrence per Orbit
- Mission Outcome Occurrences per Orbit type
  - **True Ocean**: successful landing to specific region of the ocean
  - **False Ocean**: unsuccessful landing to specific region of ocean
  - **True RTLS**: successful landing on a ground pad
  - **False RTLS**: unsuccessful landing on a ground pad
  - **True ASDS**: successful landing on a drone ship
  - **False ASDS**: unsuccessful landing on dron eship

Outcomes Labels

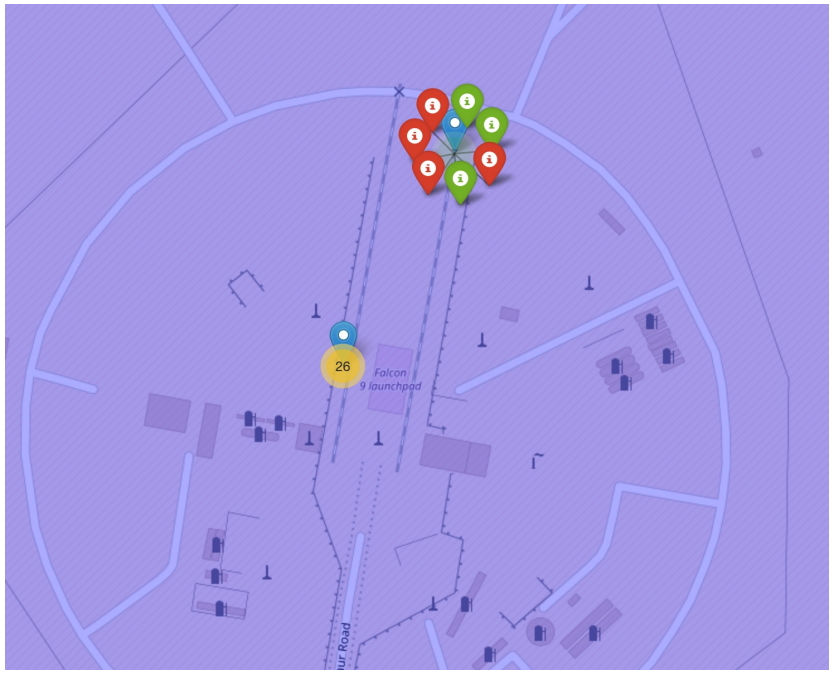
- Create list where the element is zero if the corresponding row in Outcome is in set bad_outcome
- Class labels are 2:1 success-to-failure ratio

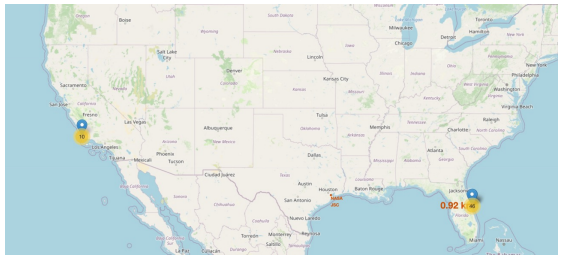
- Saved to **spacex.csv**

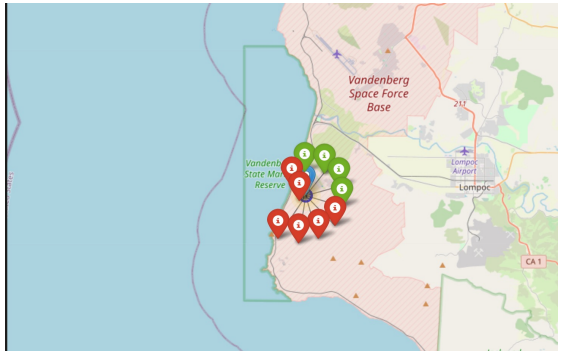
EDA with SQL

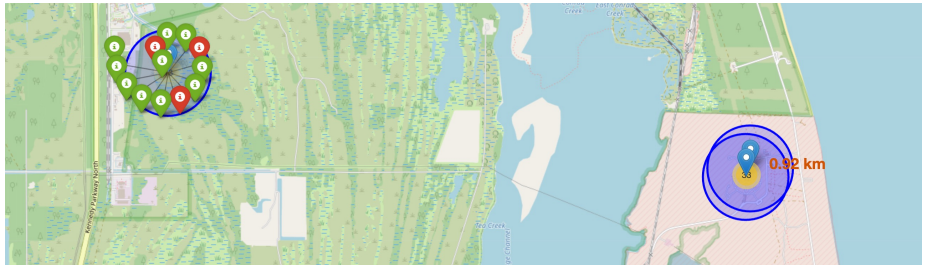
Query Process

- Stored **spacex.csv** into sqlite database
- Launch Site Names
- Display Cape Canaveral Space Launch complex 40
- Total payload mass carried by boosters launched by NASA (CRS)
- Average payload mass carried by booster version F9 v1.1
- Date when first successful ground pad landing outcome was achieved
- Successful boosters in drone ship with payload mass greater than 4000kg and less then 6000kg
- Number of successful and failure mission outcomes
- Booster Versions carrying the maximum payload mass
- 2015 failed landing outcomes in drone ship
- Count of failure (drone ship) or success (ground pad) between 2010-06-04 and 2017-03-20









Folium Analysis

- Interactive Folium maps, red for all launch sites coordinates, blue for NASA Johnson Space Center
- Green colored success markers, red unsuccessful markers
- Added coastal proximity measurement
- While launch success rate may depend on factors like payload mass and orbit type, it may also depend on location and launch site proximities – the initial position of rocket trajectories
- Building this map further justifies the use of location as a feature

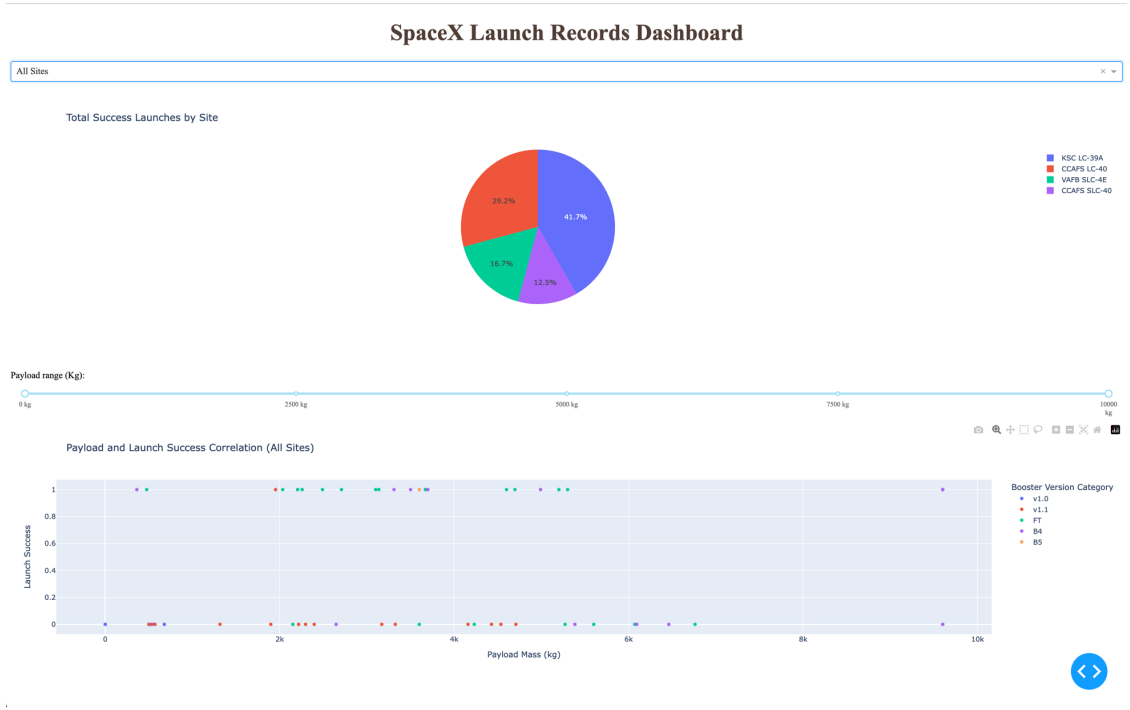
Plotly Dashboard Creation

Successful launches pie chart

- Shows total successful launches count for all sites
- Drop down selector for success focus on specific site

Payload and launch success scatter chart

- Shows success rate of different booster versions for all or certain sites
- User can select payload range
- Can focus on specific booster version if desired



EDA Visualization

Relationship Plots

- Flight Number vs Launch Site
- Payload Mass and Launch Site
- Success rate of each Orbit type
- Flight Number and Orbit type
- Payload and Orbit type

Time Series Analysis

- Number of SpaceX Launches per Year
- Average Payload Mass per Year
- Reused vs. Non-reused Launches per Year
- Reused vs. Non-Reused Boosters Success Rate

Correlation Analysis

- Correlation matrix of numeric features
- Extended correlation matrix, including categorical feature analysis
- Extended matrix converted to h-bar plot

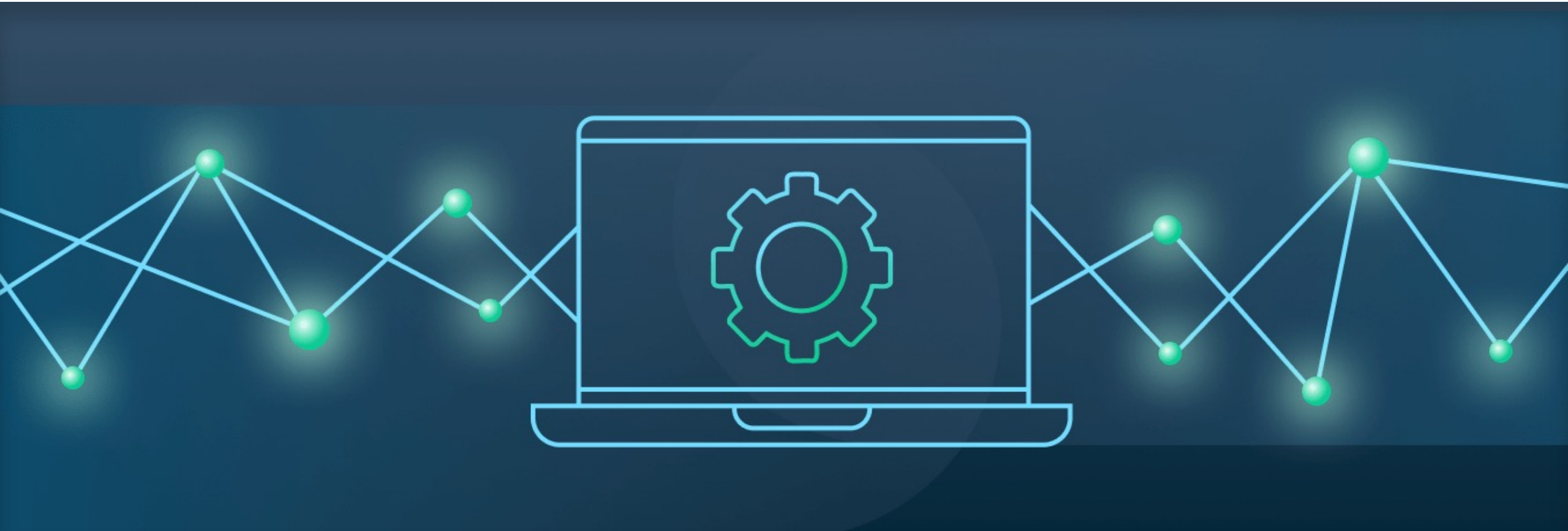
Features Engineering

Decided Features

- FlightNumber, PayloadMass, Orbit, LaunchSite, GridFins, Reused, Legs, LandingPad, Block, ReusedCount, Serial

Basic Features Engineering Process

- Create dummy variables to categorical columns, applying **OneHotEncoder** to column Orbits, LaunchSite, LandingPad, and Serial



Predictive Modeling

Preprocessing

- Set spacex_features.csv to X variable
- Create NumPy array from spacex.csv Class column to Y variable
- Standardize X with StadardScaler – fit and transform.
- Split data 20/80 using tran_test_split

Four Basic Models

- Logistic Regression, Support Vector Machine, Decision Trees, K-Nearest Neighbors
- Each using GridSearchCV with respective parameters, cv of 10, and accuracy scoring

Four Improved Models

- Improved tuning for above models, using modified GiridSearchCV parameters
- Assessed the confusion matrices, and average scores for all models
- Calculated matrix metrics, set to dataframe and visualized

Results

Results Summary

Exploratory Data Analysis

- Notable increase in launch count and reusability ratios per year
- KSC LC-39A and CCAFS LC-40 boast the highest success rates.
- The orbits designated as ES-L1, GEO, HEO, and SSO have achieved a 100% success rate.

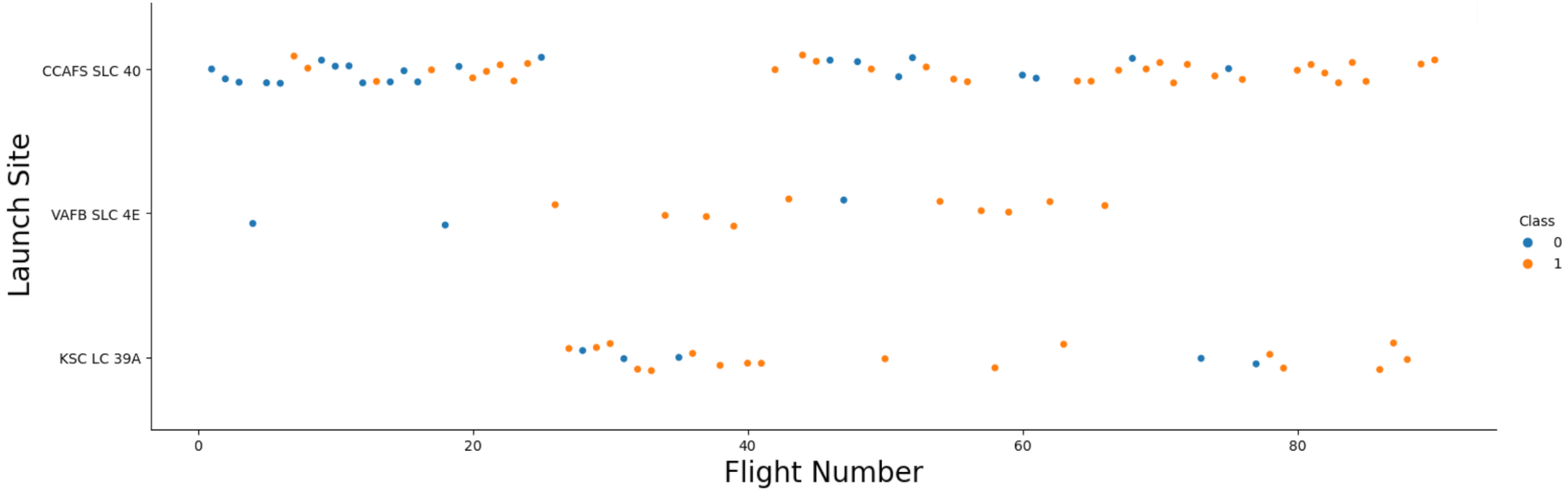
Visual Analytics

- Most launch sites are positioned near the equator, with each site situated near the coast.
- SpaceX accounts for distance to public transportation means – while this insights assisted in feature finding (Location, Launch Site)
- Successful launch site markers in conjunction with time series analysis in an interesting combination – a 'face' to findings

Predictive Analytics

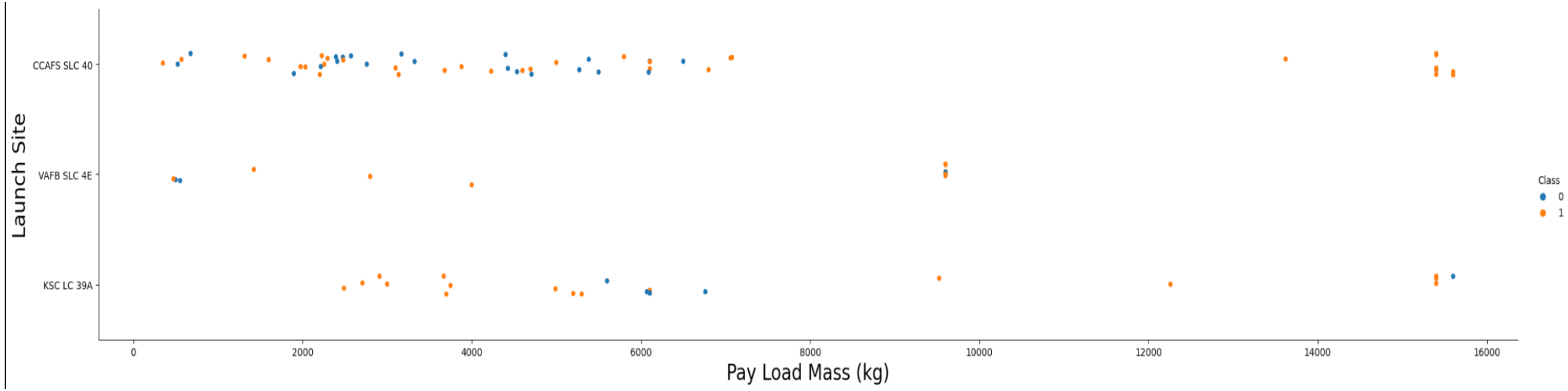
- The Decision Tree 2 and Logistic Regression 2 model vie for most accurate model


EDA Insights



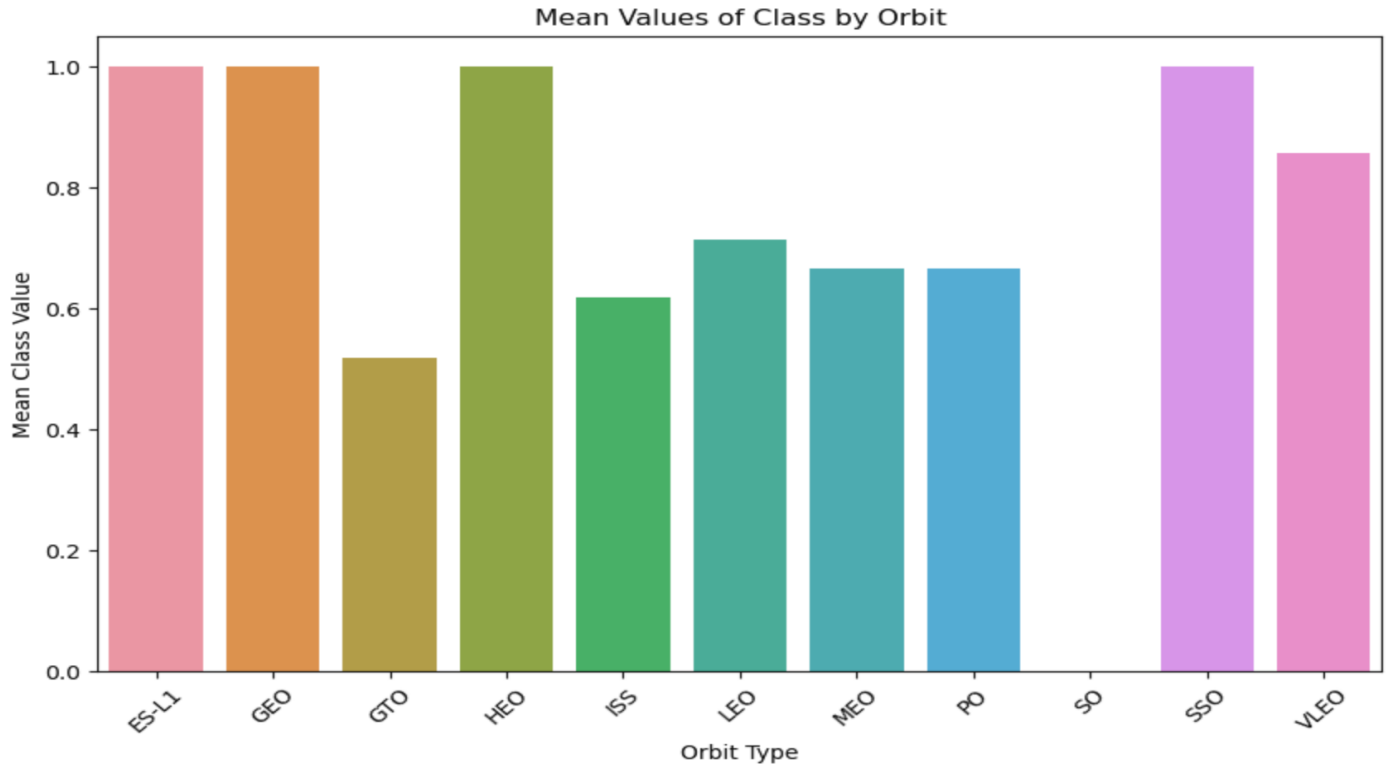
Flight Number and Launch Site

- Majority of launches took place from CCAFS SLC-40, with a temporary focus on KSC LC 39A
- KSC LC 39A launch frequency increased over time, especially recently
- This chart leads to Payload and Launch Site scatter graph creation



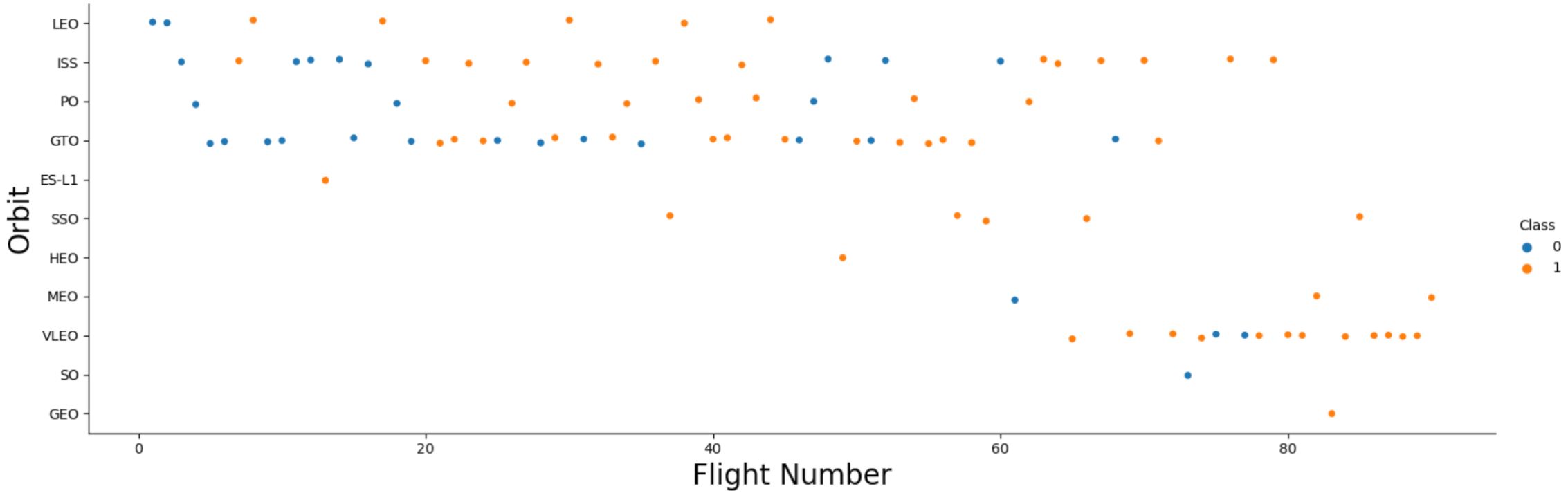
Payload and Launch Site

- Most launches are successful across payload masses
- If failures are not clustered around certain payload mass ranges, then failures may not be heavily dependent only payload mass
- Differences in launch site success rate possibly indicate differences in launch site infrastructure and operational practices – this can be explored for advanced model tuning



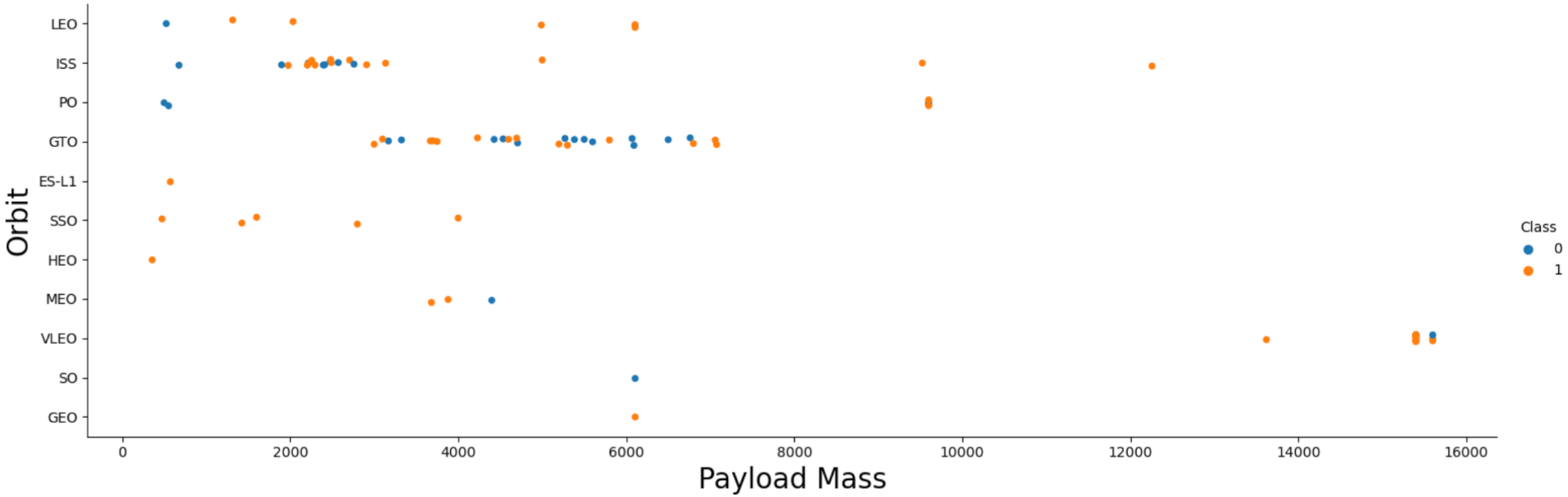
Mean Class Values by Orbit

- ES-L1, GEO, LEO, HEO, and VLEO are high success orbits
- GTO, ISS, HEO, and PO are mixed success orbits – higher risk missions, different orbit parameters? More complex factors at play
- Affirms the Falong's success in diverse orbits, and justifies Orbits as a feature



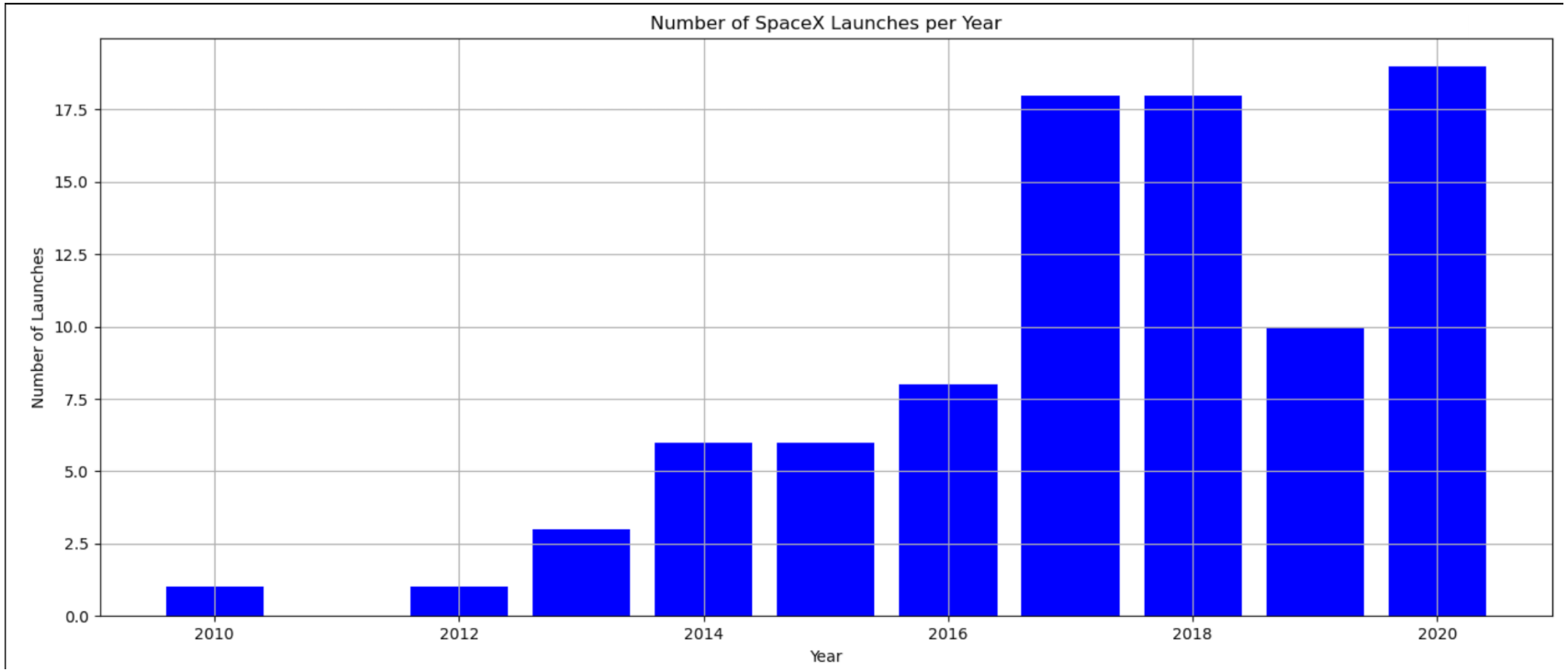
Flight Number and Orbit

- In LEO orbit, success appears to be related to the number of flights
- There seems to be no relationship between flight number when in GTO orbit
- Very few flights are directed toward VELO, MEO, HEO, ES-L1 are low frequency flights



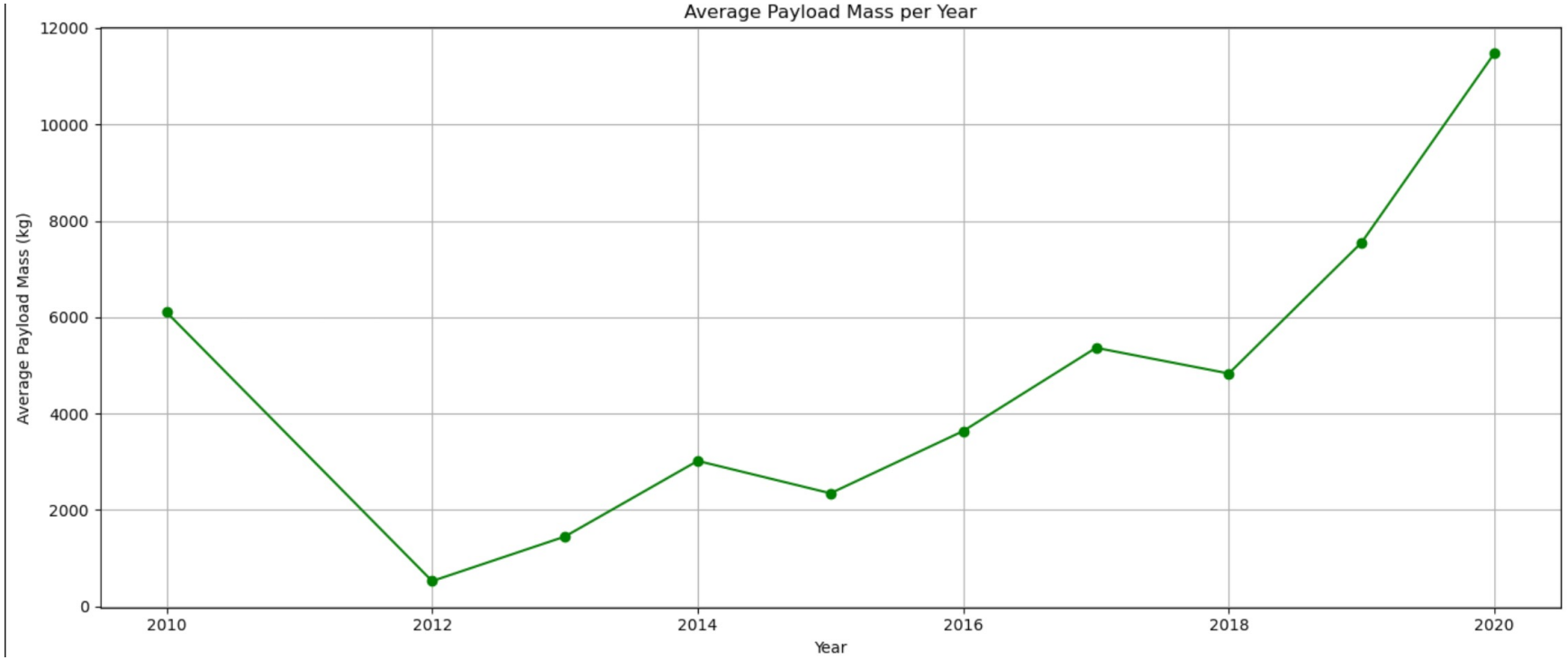
Payload Mass and Orbit

- With heavy payloads, successful landing or positive landing rates are more for Polar, LEO, and ISS
- For GTO, we cannot distinguish for both 0 and 1's are present, from ~3000kg to ~7000kg
- Payload Mass correlation – a possible tendency for increased failure rates with increasing payload mass



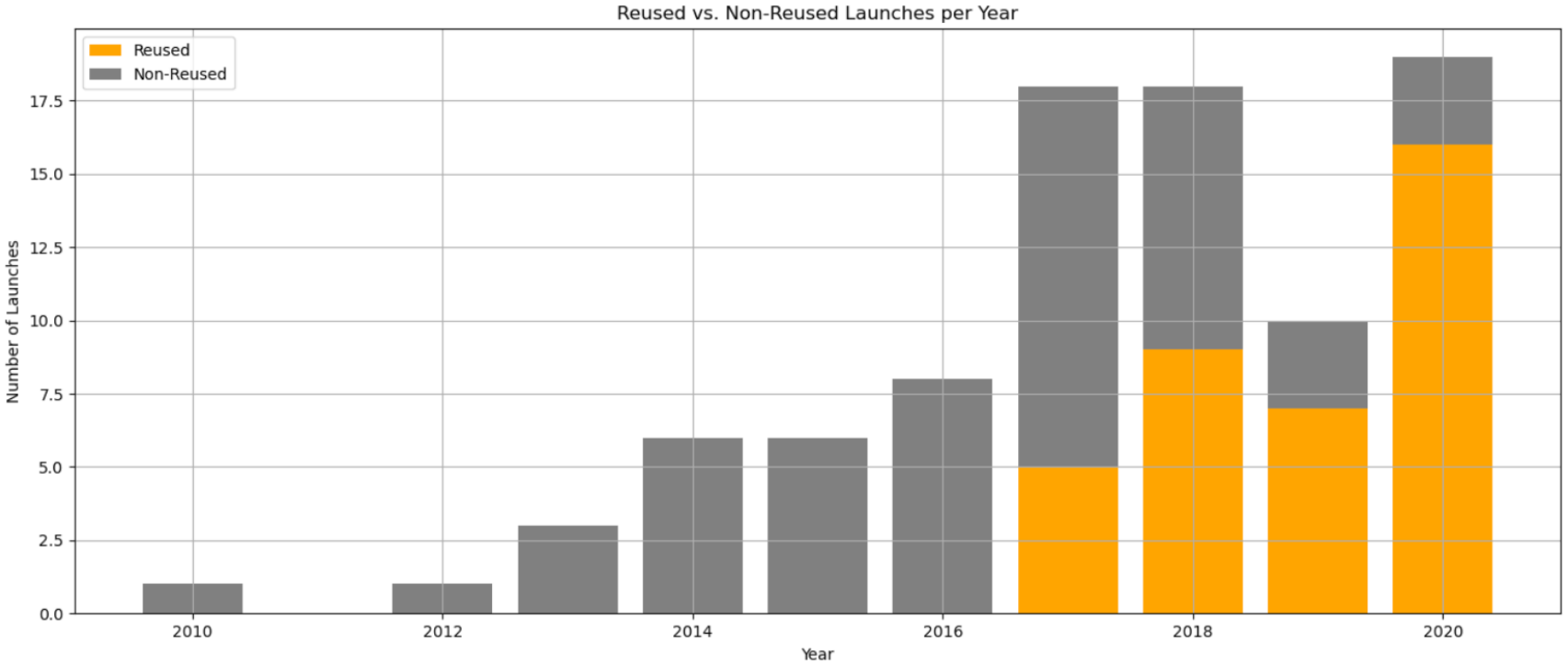
Number of SpaceX Launches per Year

- Trend from 2016 onward displays a peak in SpaceX's operational tempo – indicates growth, operational milestones



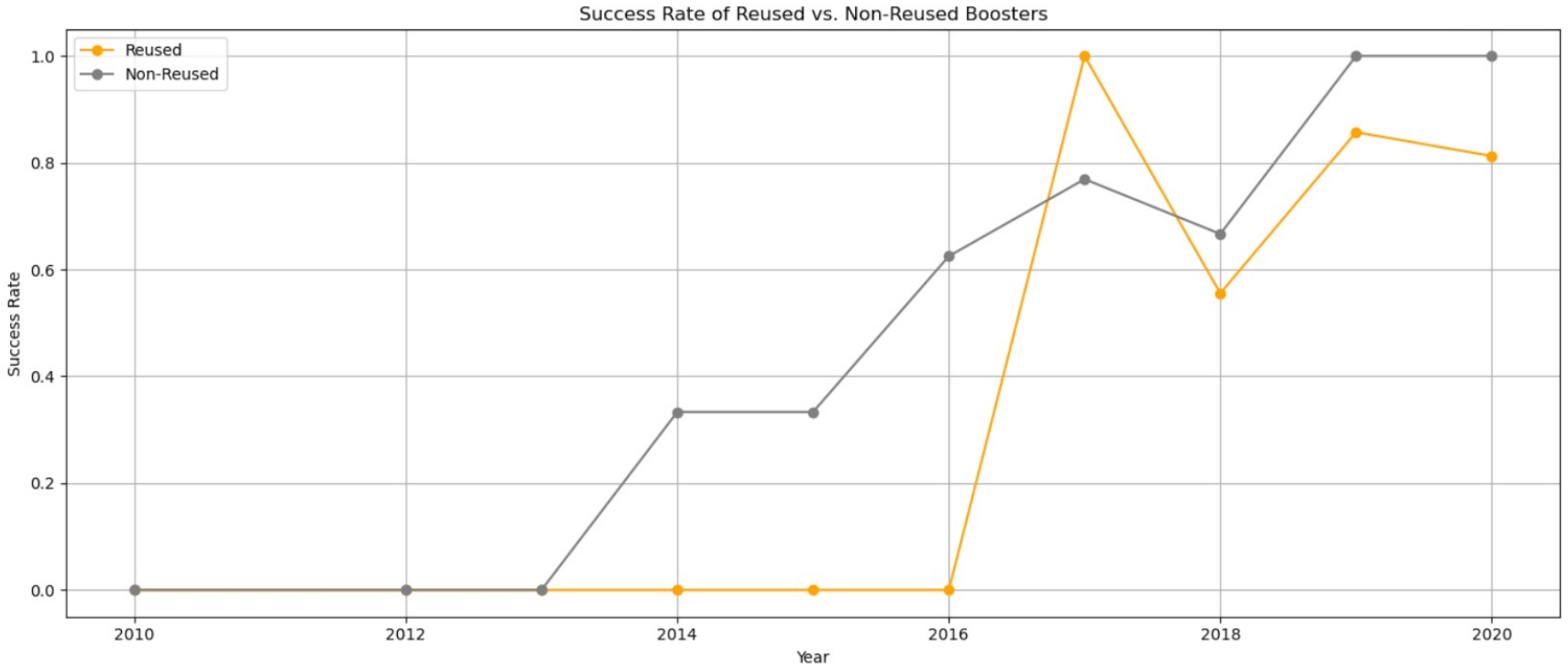
Average Payload Mass per Year

- Progressive rise since 2012 alludes to low-risk payload masks as launch capabilities were experimented with
- Will include payload mass as a continuous variable



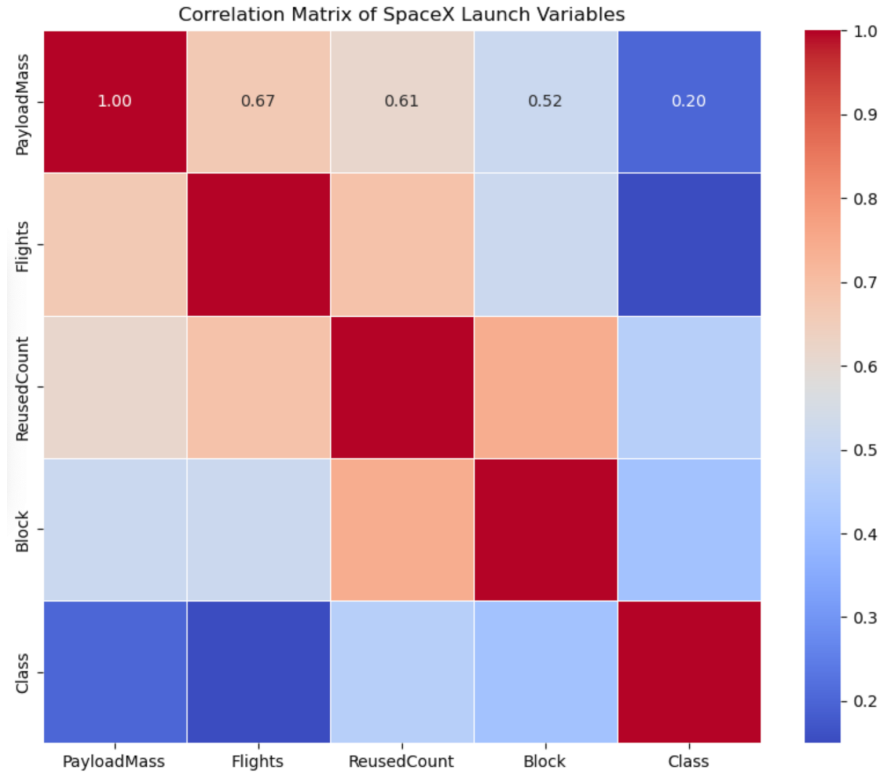
Reused vs Non-Reused Launches per Year

- Reusability ratios positively increases post 2016
- Reusability factor will be a significant predictor for the models



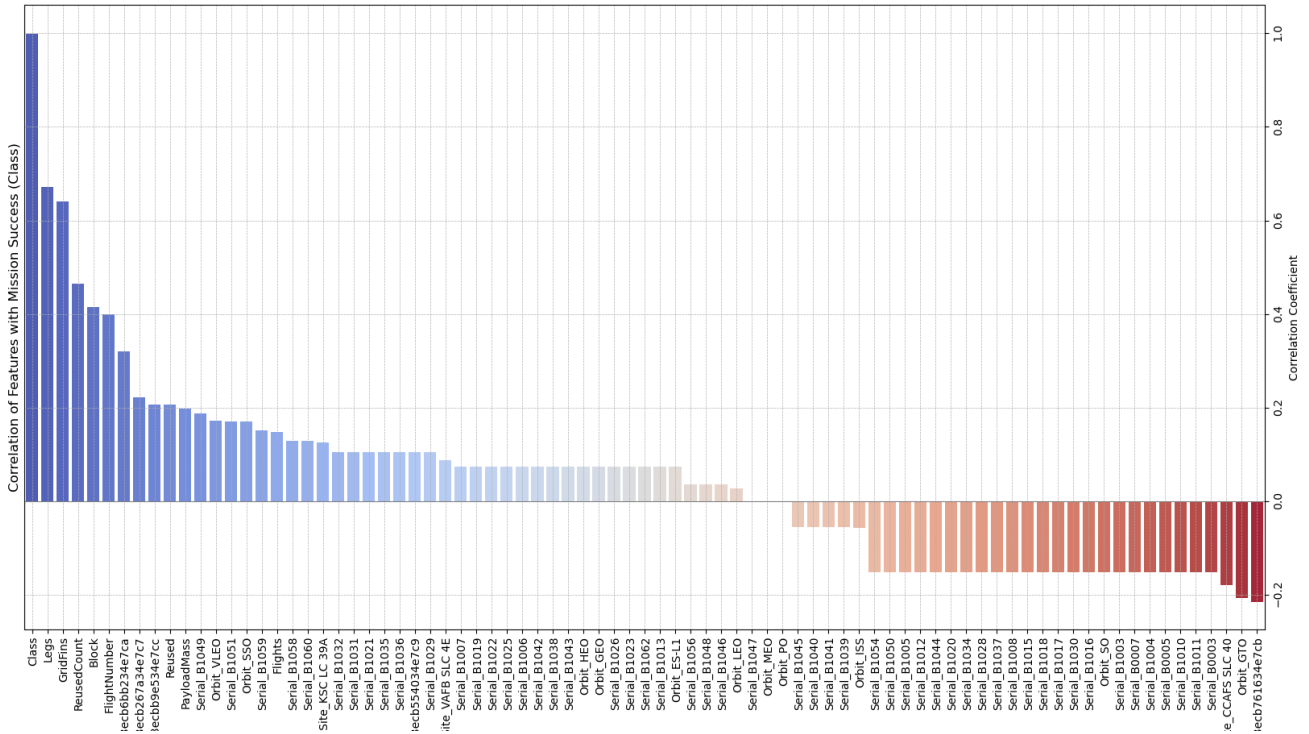
Success Rate of Reused vs. Non-Reused Boosters

- Stabilizing success rate variance in model creation will be important
- Success rates of Reused and Non-Reused seem to mirror each other post 2018



Correlation Analysis

- There are positive correlations between PayloadMass/Fights/ ReusedCount/Block and Class albeit low (~0.20 each)
- Despite this, they have "sort-of" strong correlations with each other
  - PayloadMass and Flights (0.67)
  - PayloadMass and ReusedCount(0.61)
  - PayloadMass and Block (0.52)
  - Flights and ReusedCount (0.52)



Correlation of Features with Mission Success (Class)

Positive Features

- Landing Legs, GridFins, ReusedCount, Block, FlightNumber, Payload Mass, Certain newer/specific Booster Serials, Certain Orbits

Negative Features

- Certain Orbits, older/specific Booster Serials

SQL Queries 1-2

- Show Unique Launch Sites
  - Uses DISTINCT
- Select first five Cape Canaveral Launch Sites
  - Uses WHERE and LIKE

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

[32] ✓ 0.0s

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

[44] ✓ 0.0s Python

```
* sqlite:///my_data1.db
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	7:44:00	F9 v1.0 B0005	CCAFS LC-40	Dragon demo flight C2	525	LEO (ISS)	NASA (COTS)	Success	No attempt
2012-10-08	0:35:00	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

```
1  %%sql SELECT SUM(PAYLOAD_MASS__KG_)
2  AS Total_Payload_Mass
3  FROM SPACEXTBL
4  WHERE Customer = 'NASA (CRS)';
```

✓ 0.0s

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

Total_Payload_Mass
45596

```
1  %%sql SELECT AVG(PAYLOAD_MASS__KG_)
2  AS Avg_Payload_Mass
3  FROM SPACEXTBL
4  WHERE Booster_Version = 'F9 v1.1';
```

✓ 0.0s

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

Avg_Payload_Mass
2928.4

SQL Queries 3-4

- Display the total payload mass carried by boosters launched by NASA (CRS)
  - Uses SUM, AS, WHERE
- Display average payload mass carried y booster version F9 v1.1
  - Uses AVG, AS, FROM

```
%%sql SELECT Booster_Version
FROM SPACEXTBL
WHERE PAYLOAD_MASS__KG_ > 4000 AND PAYLOAD_MASS__KG_ < 6000
AND Landing_Outcome = 'Success (drone ship)';
```

✓ 0.0s

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

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

```
%%sql SELECT MIN(Date) AS first_successful_ground
FROM SPACEXTBL
WHERE Landing_Outcome = 'Success (ground pad)';
```

✓ 0.0s

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

first_successful_ground
2015-12-22

SQL Queries 5-6

- Lists the data when the first successful landing outcome in ground pad was achieved
  - Uses MIN, WHERE
- Lists the names of the boosters which have success in drone ship and have payload mass greater than 4000kg but less than 6000kg
  - Uses WHERE, >, <, AND

```
%%sql SELECT Mission_Outcome, COUNT(*) AS Total_Count
FROM SPACEXTBL
GROUP BY Mission_Outcome
```

✓ 0.0s

* sqlite:///my_data1.db
Done.

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

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

✓ 0.0s

* sqlite:///my_data1.db
Done.

Booster_Version	PAYLOAD_MASS__KG_
F9 B5 B1048.4	15600
F9 B5 B1049.4	15600
F9 B5 B1051.3	15600
F9 B5 B1056.4	15600
F9 B5 B1048.5	15600
F9 B5 B1051.4	15600
F9 B5 B1049.5	15600
F9 B5 B1060.2	15600
F9 B5 B1058.3	15600
F9 B5 B1051.6	15600
F9 B5 B1060.3	15600
F9 B5 B1049.7	15600

SQL Queries 7-8

- Lists the total number of successful and failure mission outcomes
  - Uses COUNT, AS, GROUP BY
- Lists the names of the booster_versions which have carried the maximum payload mass
  - Uses WHERE w/ subquery, MAX

```
%%sql
SELECT
    Booster_Version,
    Launch_Site,
    substr(Date, 6, 2) AS Month,
    Landing_Outcome
FROM SPACEXTBL
WHERE
    substr(Date, 0, 5) = '2015'
    AND Landing_Outcome = 'Failure (drone ship)';
```

✓ 0.0s

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

Booster_Version	Launch_Site	Month	Landing_Outcome
F9 v1.1 B1012	CCAFS LC-40	01	Failure (drone ship)
F9 v1.1 B1015	CCAFS LC-40	04	Failure (drone ship)

SQL Queries 9-10

```
%%sql
SELECT Landing_Outcome, COUNT(*) AS Outcome_Count
FROM SPACEXTBL
WHERE (Date BETWEEN '2010-06-04' AND '2017-03-20')
GROUP BY Landing_Outcome
ORDER BY Outcome_Count DESC;
```

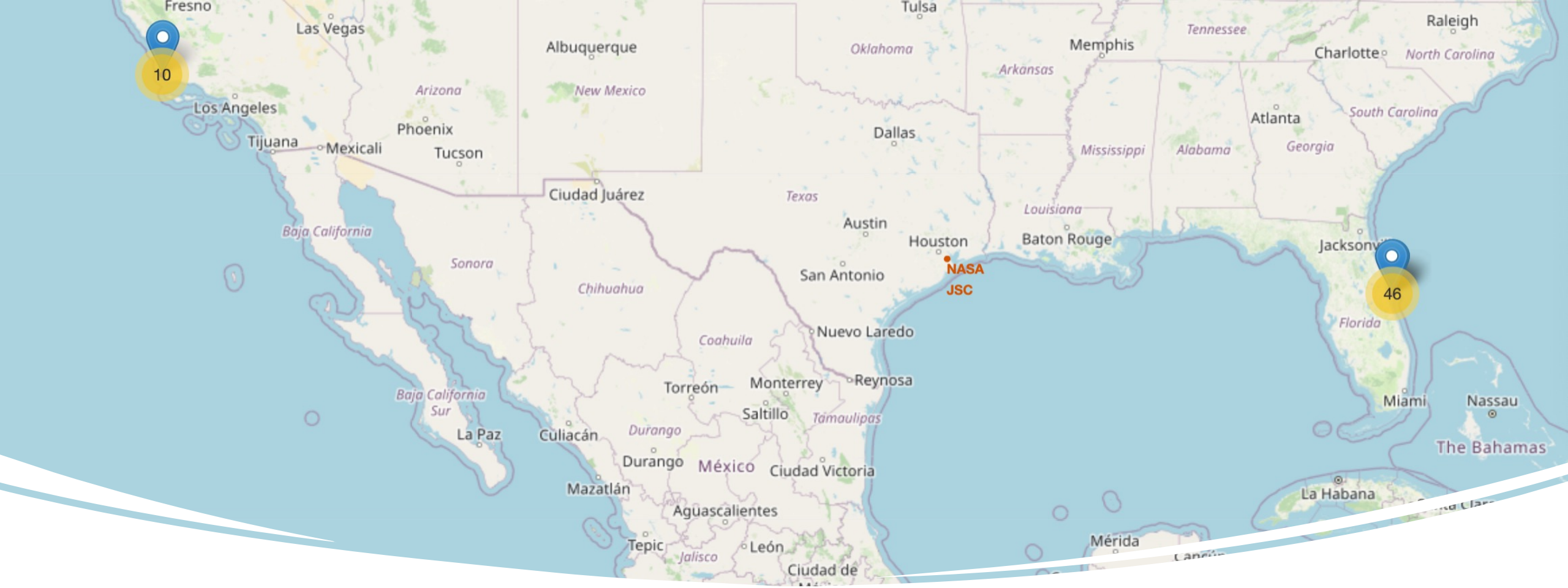
✓ 0.0s

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

Landing_Outcome	Outcome_Count
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

- List the records which will display the month names, failure landing_outcomes in drone ship, booster_versions, launch_site for the months in 2015
  - Uses AS, WHERE, SUBSTR, AND
- Rank the count of landing outcomes between 2010-06-04 and 2017-03-20
  - Uses COUNT, WHERE, BETWEEN, AND, GROUP BY, ORDER BY

Folium Analysis

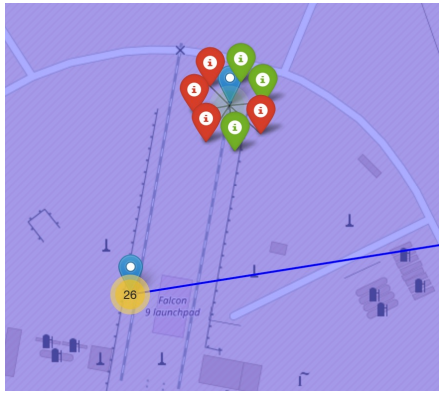


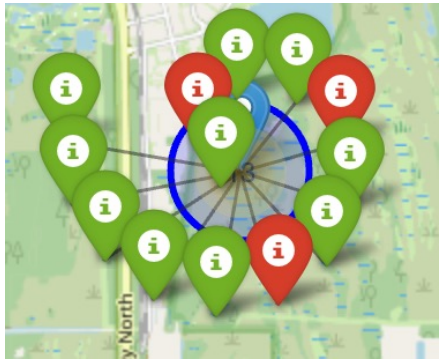
Folium Map

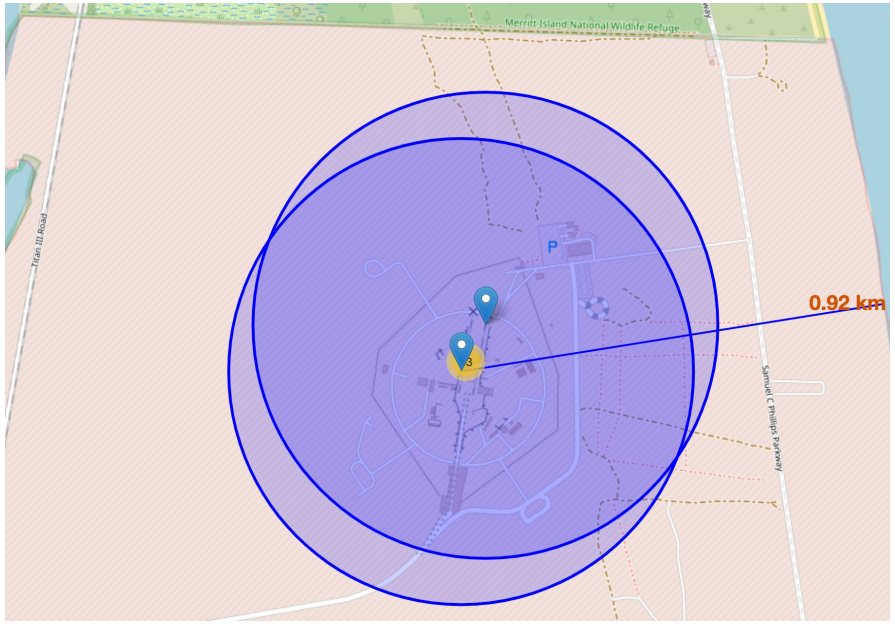
- This Folium map with markers denotes the two primary regions for launches
- Vandenberg Air Force Base hosts the western launch site
- The eastern marker points to Cape Canaveral Air Force Station and Kennedy Space Center
- Quick research affirms equatorial range and coastal proximity as primary geographical drivers of launch site location

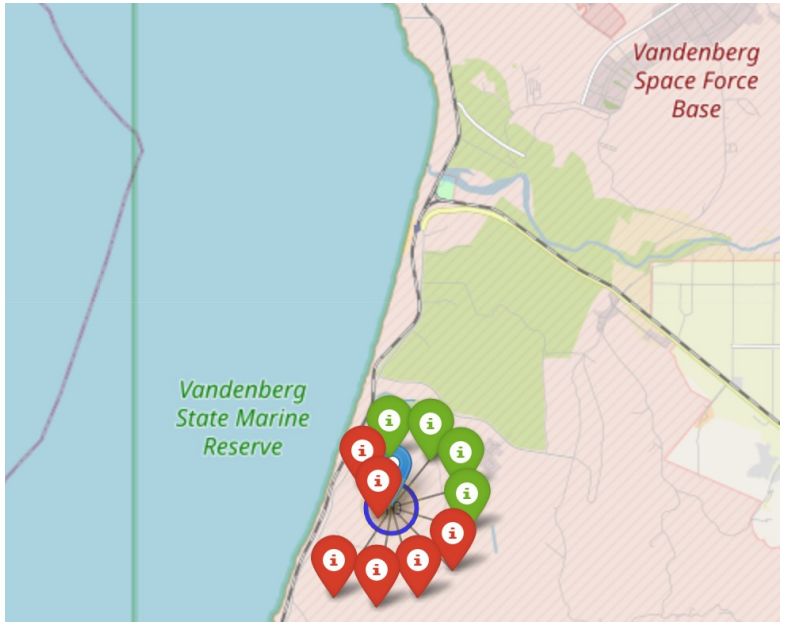
Launch Site Map Locations

Clockwise – CCAFS SLC-40, CCAFS Coastal Proximity, KSC LC-39A, CCAFS LC-40, VAFB SLC-4E



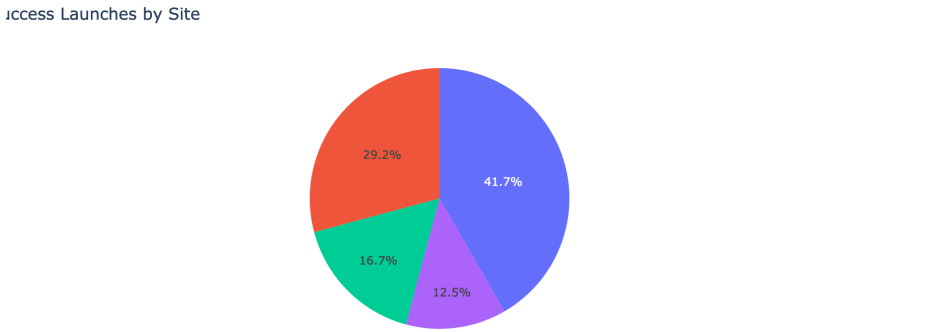


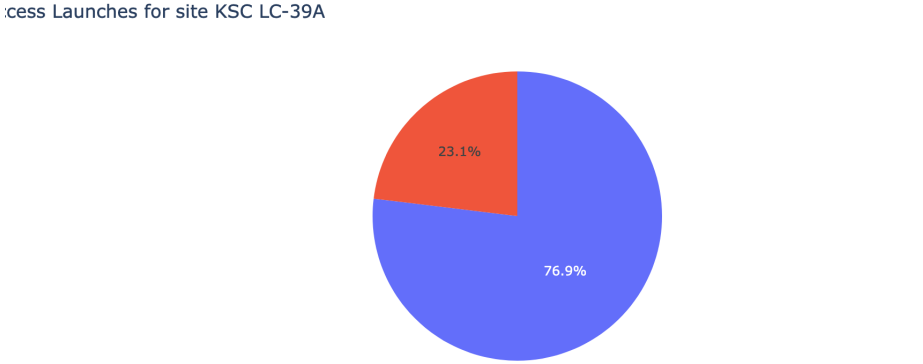




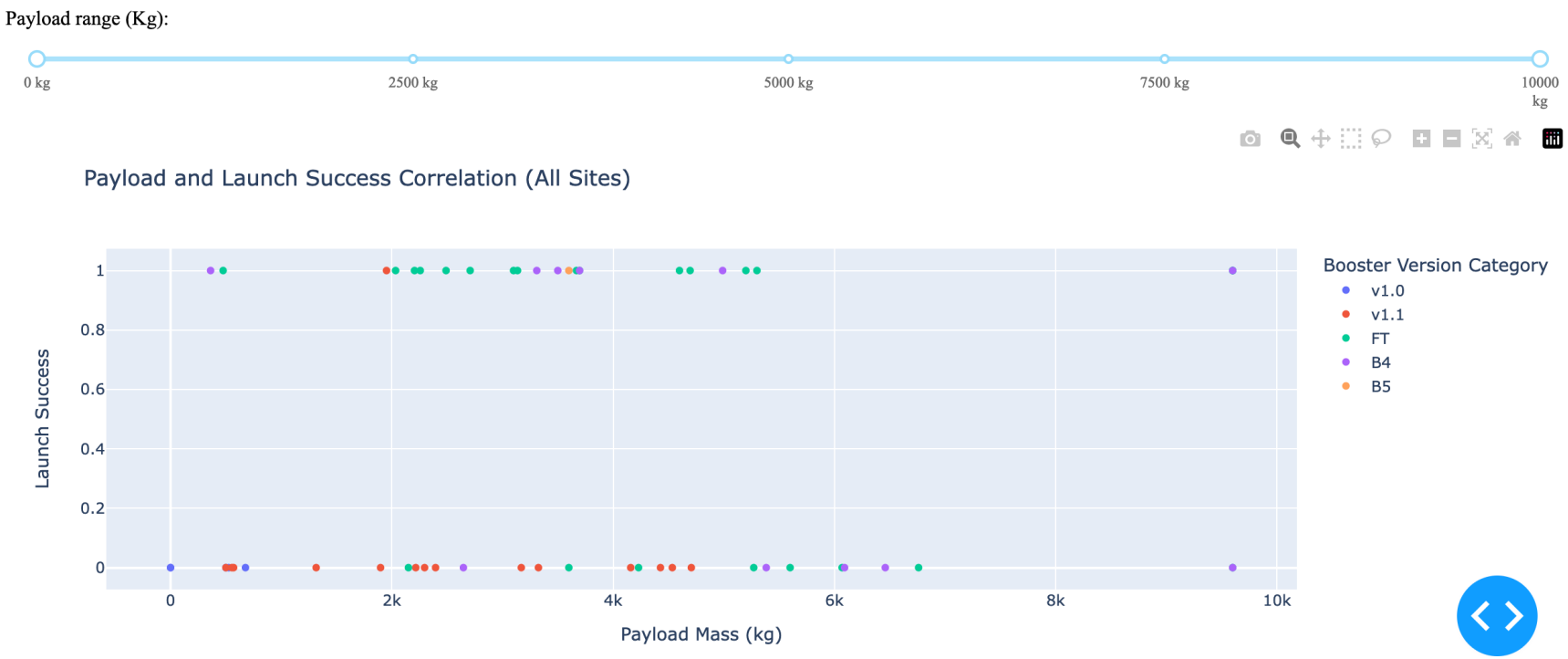

Dashboard Insights

Pie Chart Insights





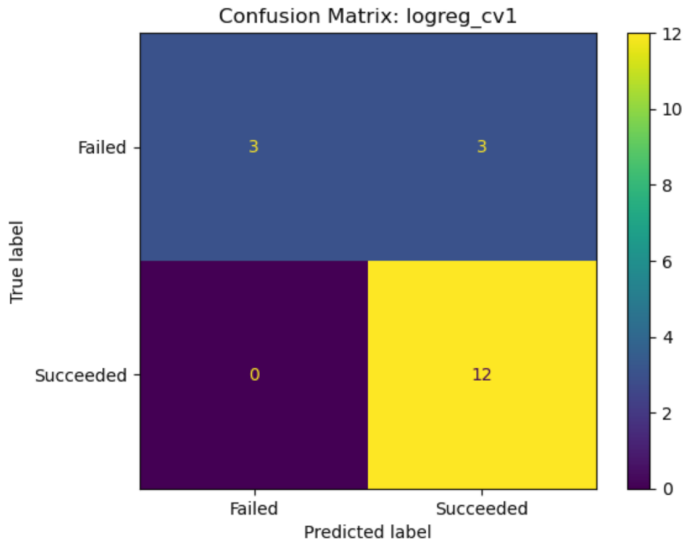
- Kennedy Space Center Site had the most amount of launch sites
- It also leads in the best success-to-failure ratio
- Our correlation analysis supports these positive metrics

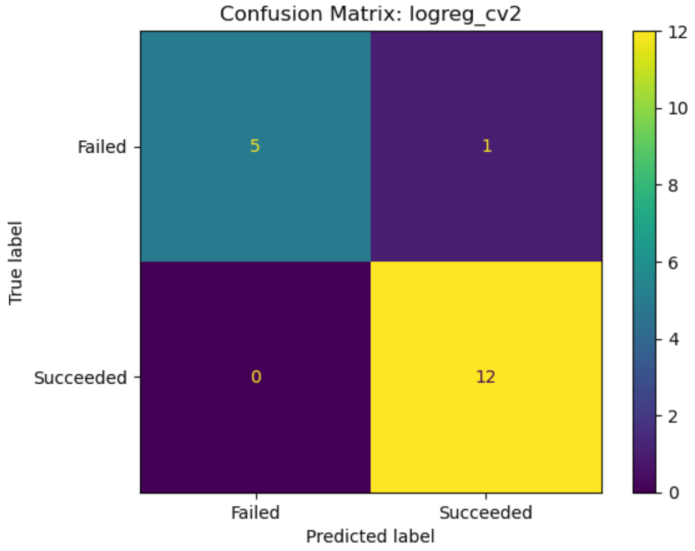


Scattergraph Insights

- Newer booster version have higher success rates, particularly B4 and B5
- Payload range 1900-5300kg has the highest launch success rate
- Payload ranges 500-1000kg, 2200-2700kg, 4200-4700kg, and 5200-6700kg have low success rates
- FT has the highest success rate, in payloads between 2000-4000k

Predictive Modeling






```
parameters = {"C":[0.01,0.1,1],'penalty':['l2'], 'solver':['lbfgs']}# l1 lasso l2 ridge
lr1=LogisticRegression()

lr_grid_search1 = GridSearchCV(estimator = lr1, param_grid = parameters, cv = 10, scoring = 'accuracy')

logreg_cv1 = lr_grid_search1.fit(X_train, Y_train)
logreg_cv1
```

Python

GridSearchCV
estimator: LogisticRegression
LogisticRegression

```
print("tuned hyperparameters :(best parameters) ",logreg_cv1.best_params_)
print("accuracy :",logreg_cv1.best_score_)
```

Python

```
tuned hyperparameters :(best parameters)  {'C': 1, 'penalty': 'l2', 'solver': 'lbfgs'}
accuracy : 0.8214285714285714
```

```
from sklearn.utils.class_weight import compute_class_weight

class_weights = compute_class_weight('balanced', classes=np.unique(Y), y=Y)
class_weight_dict = {i: class_weights[i] for i in range(len(class_weights))}

lr2 = LogisticRegression(class_weight=class_weight_dict)
lr_grid_search2 = GridSearchCV(estimator = lr2, param_grid = parameters, cv = 10, scoring='roc_auc')

logreg_cv2 = lr_grid_search2.fit(X_train, Y_train)
logreg_cv2
```

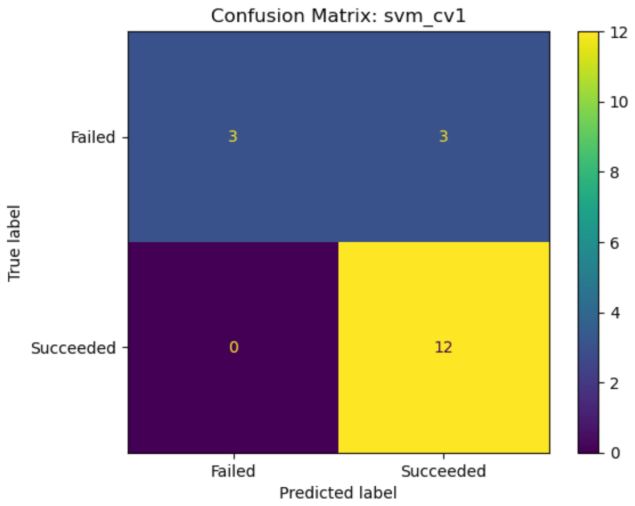
GridSearchCV
estimator: LogisticRegression
LogisticRegression

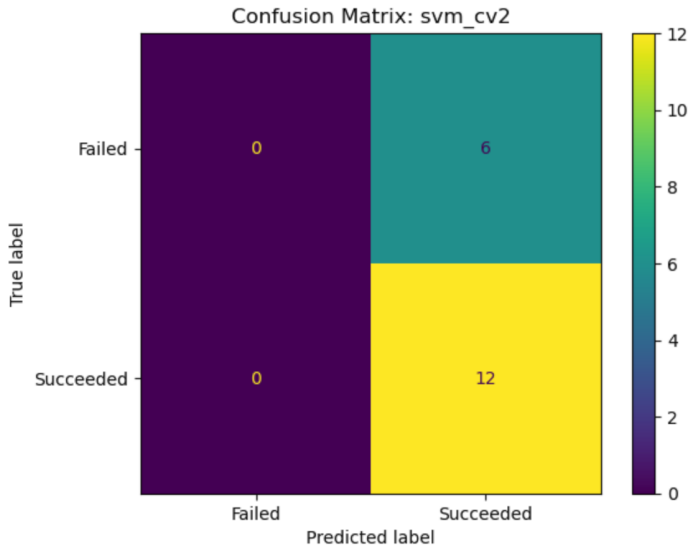
```
print("tuned hyperparameters :(best parameters) ",logreg_cv2.best_params_)
print("accuracy :",logreg_cv2.best_score_)
```

```
tuned hyperparameters :(best parameters)  {'C': 0.01, 'penalty': 'l2', 'solver': 'lbfgs'}
accuracy : 0.8716666666666667
```

Logistic Regression

- Class imbalance (2:1) necessitates being mindful of any model's sensitivity to the target variable distribution
- Switched performance metric from accuracy to AUC-ROC to mitigate this
- Added penalty weights class_weights to 'balanced' adjusts the weights inversely proportional to class frequencies






```
parameters = {'kernel':('linear', 'rbf','poly','rbf', 'sigmoid'),
              'C': np.logspace(-3, 3, 5),
              'gamma':np.logspace(-3, 3, 5)}
svm1 = SVC()
```

```
svm_grid_search1 = GridSearchCV(estimator=svm1, param_grid=parameters, cv=10, scoring='accuracy')

svm_cv1 = svm_grid_search1.fit(X_train, Y_train)
svm_cv1
```

GridSearchCV
estimator: SVC
SVC

```
print("tuned hpyerparameters :(best parameters) ",svm_cv1.best_params_)
print("accuracy :",svm_cv1.best_score_)
```

```
tuned hpyerparameters :(best parameters)  {'C': 1.0, 'gamma': 0.03162277660168379, 'kernel': 'sigmoid'}
accuracy : 0.8482142857142858
```

```
parameters = {
    'kernel': ['linear', 'rbf', 'poly', 'sigmoid'],
    'C': np.logspace(-4, 4, 10),
    'gamma': np.logspace(-4, 4, 10),
    'degree': [2, 3, 4]
}

svm2=SVC()
```

```
parameters = {
    'kernel': ['linear', 'rbf', 'poly', 'sigmoid'],
    'C': np.logspace(-4, 4, 10),
    'gamma': np.logspace(-4, 4, 10),
    'degree': [2, 3, 4]
}

svm2=SVC()
```

```
svm_grid_search2 = GridSearchCV(estimator=svm2, param_grid=parameters, cv=10, scoring='roc_auc')

svm_cv2 = svm_grid_search2.fit(X_train, Y_train)
svm_cv2

```

GridSearchCV
estimator: SVC
SVC

```
print("tuned hpyerparameters :(best parameters) ",svm_cv2.best_params_)
print("accuracy :",svm_cv2.best_score_)
```

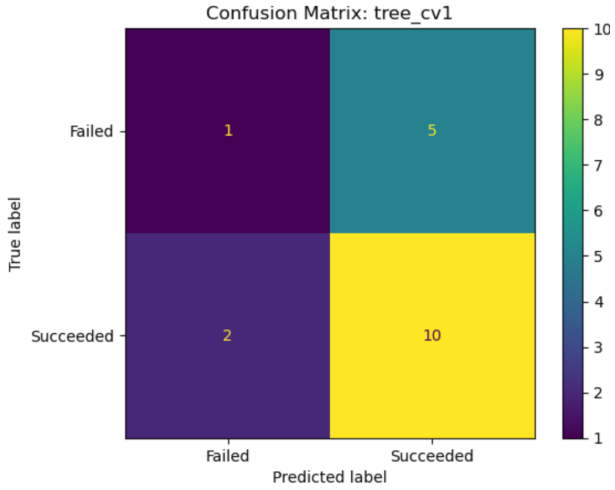
```
tuned hpyerparameters :(best parameters)  {'C': 0.005994842503189409, 'degree': 2, 'gamma': 0.0007742636826811277, 'kernel': 'rbf'}
accuracy : 0.9
```

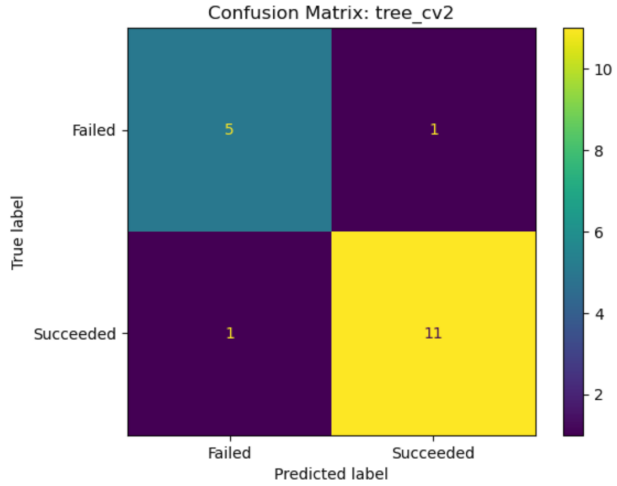
Support Vector Machine

- Increased logspace in the second model
- Accuracy` is straightforward, but does not necessarily reflect nuances of imbalanced datasets
- Second model produced the most false-positives out of the eight

Decision Trees

- Model 1, like Model 1 for of the before models, prioritizes overall accuracy, where Model 2 uses 'class_weight' as 'balanced to that both classes receive the same importance
- Model 2 has more freedom to use all available features for decision making





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

tree1 = DecisionTreeClassifier()
```

Python

```
tree_grid_search1 = GridSearchCV(estimator=tree1, param_grid=parameters, cv=10, scoring='accuracy', error_score='raise')
tree_cv1 = tree_grid_search1.fit(X_train, Y_train)
tree_cv1
```

Python

GridSearchCV
estimator: DecisionTreeClassifier
DecisionTreeClassifier

```
print("tuned hyperparameters :(best parameters) ",tree_cv1.best_params_)
print("accuracy :",tree_cv1.best_score_)
```

Python

```
tuned hyperparameters :(best parameters)  {'criterion': 'gini', 'max_depth': 6, 'max_features': 'sqrt', 'min_samples_leaf': 1, 'min_samples_split': 10, 'splitter': 'random'}
accuracy : 0.8482142857142856
```

```
parameters = {'criterion': ['gini', 'entropy'],
    'splitter': ['best', 'random'],
    'max_depth': [2*n for n in range(1,10)],
    'max_features': ['sqrt', 'log2', None],
    'min_samples_leaf': [1, 2, 4],
    'min_samples_split': [2, 5, 10],
    'class_weight': ['balanced'],
    'ccp_alpha': [0.0, 0.01, 0.1]}

tree2 = DecisionTreeClassifier()
```

Python

```
strat_cv = StratifiedKFold(n_splits=10)
tree_grid_search2 = GridSearchCV(estimator=tree2, param_grid=parameters, cv=strat_cv, scoring='roc_auc')
tree_cv2 = tree_grid_search2.fit(X_train, Y_train)
tree_cv2
```

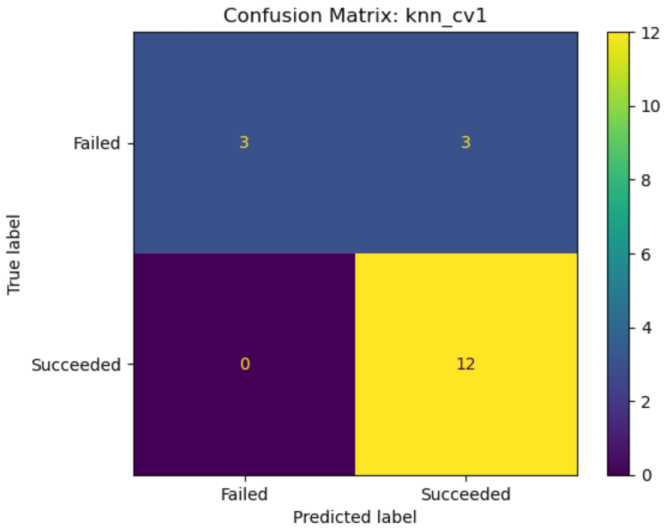
Python

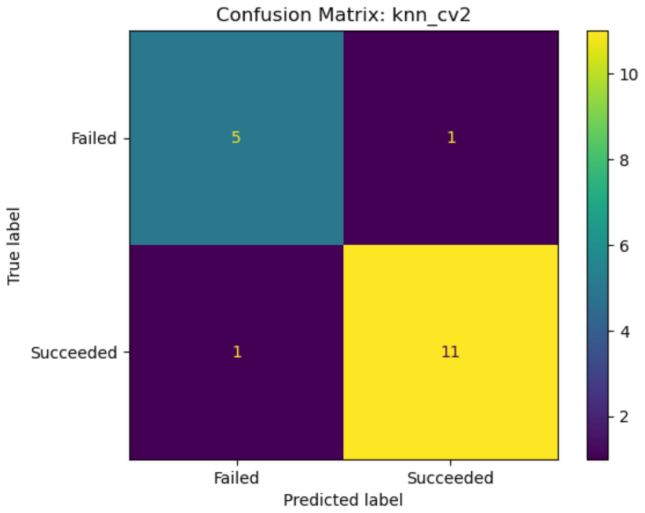
GridSearchCV
estimator: DecisionTreeClassifier
DecisionTreeClassifier

```
print("tuned hyperparameters :(best parameters) ",tree_cv2.best_params_)
print("score :",tree_cv2.best_score_)
```

Python

```
tuned hyperparameters :(best parameters)  {'ccp_alpha': 0.0, 'class_weight': 'balanced', 'criterion': 'gini', 'max_depth': 2, 'max_features': None, 'min_samples_leaf': 1, 'min_samples_split':
score : 0.9008333333333333
```






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

KNN1 = KNeighborsClassifier()
```

```
knn_grid_search1 = GridSearchCV(estimator=KNN1, param_grid=parameters, cv=10, scoring='accuracy', error_score='raise', n_jobs = 1)
knn_cv1 = knn_grid_search1.fit(X_train, Y_train)
knn_cv1
```

GridSearchCV
estimator: KNeighborsClassifier
KNeighborsClassifier

```
print("tuned hyperparameters :(best parameters) ",knn_cv1.best_params_)
print("accuracy :",knn_cv1.best_score_)
```

```
tuned hyperparameters :(best parameters)  {'algorithm': 'auto', 'n_neighbors': 6, 'p': 1}
accuracy : 0.8339285714285714
```

```
from imblearn.over_sampling import SMOTE
from imblearn.pipeline import Pipeline
```

```
pipeline = Pipeline([
    ('smote', SMOTE(random_state=42)),
    ('knn', KNeighborsClassifier(algorithm='auto'))
])

parameters = {
    'knn__n_neighbors': range(1, 20),
    'knn__weights': ['uniform', 'distance'],
    'knn__p': [1, 2]
}
```

```
knn_grid_search2 = GridSearchCV(pipeline, parameters, cv=StratifiedKFold(5), scoring='roc_auc')
knn_cv2 = knn_grid_search2.fit(X_train, Y_train)
knn_cv2
```

GridSearchCV
estimator: Pipeline
SMOTE
KNeighborsClassifier

```
print("tuned hyperparameters :(best parameters) ",knn_cv2.best_params_)
print("best cross-validation score:",knn_cv2.best_score_)
```

```
tuned hyperparameters :(best parameters)  {'knn__n_neighbors': 18, 'knn__p': 2, 'knn__weights': 'uniform'}
best cross-validation score: 0.8664444444444446
```

K-Nearest Neighbors

- Model 1 uses Manhattan distance, where Model 2 uses Euclidean distance
- Model 2 maintains ROC-AUC for scoring, and addresses class imbalance with SMOTE
- Model 2 created considering the complexity of the dataset – this model takes the longest to run

Best Model Determination

- Calculated best score
- Decision Tree 2 computed as best model

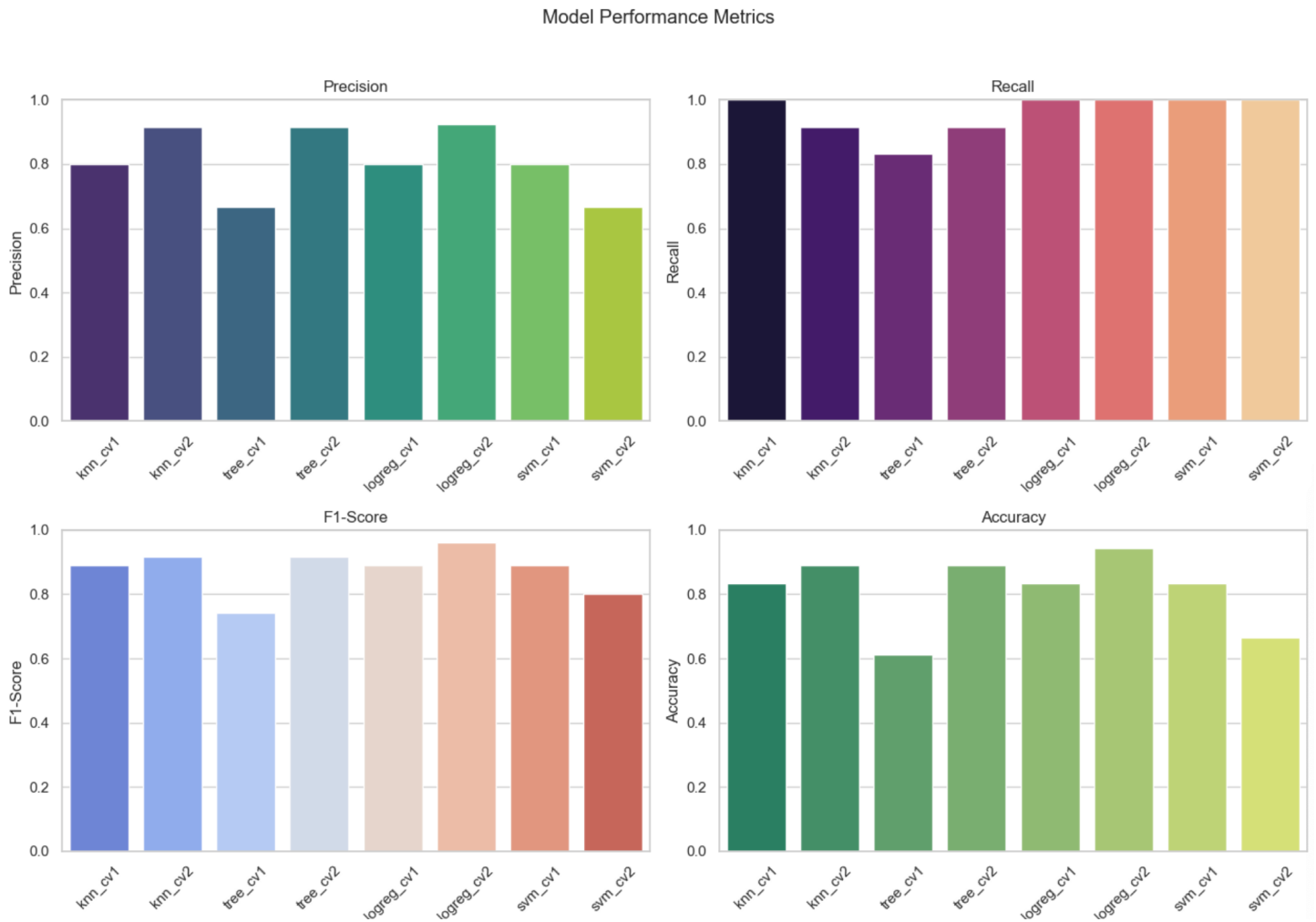
```
best_model = max(models, key=lambda x: models[x]['score'])
best_score = models[best_model]['score']
best_params = models[best_model]['params']


print(f"The best model is {best_model} with a score of {best_score}")
print("Best parameters:", best_params)
```

```
The best model is DecisionTree2 with a score of 0.9008333333333333
Best parameters: {'ccp_alpha': 0.0, 'class_weight': 'balanced', 'criterion':
```

- Calculated confusion matrix scores
- Logistic Regression 2 arguably best model

	Precision	Recall	F1-Score	Accuracy
knn_cv1	0.800000	1.000000	0.888889	0.833333
knn_cv2	0.916667	0.916667	0.916667	0.888889
tree_cv1	0.666667	0.833333	0.740741	0.611111
tree_cv2	0.916667	0.916667	0.916667	0.888889
logreg_cv1	0.800000	1.000000	0.888889	0.833333
logreg_cv2	0.923077	1.000000	0.960000	0.944444
svm_cv1	0.800000	1.000000	0.888889	0.833333
svm_cv2	0.666667	1.000000	0.800000	0.666667

Model Performance Metrics



Best Model Discussion

- Considering the business problem, where SpaceY is looking to create bids to facilitate cost determination for Falcon9 launches, it is critical that we prioritize avoiding false positives.
- False positive avoidance may entail models with higher precision scores
  - However, high performing models might mean model overfitting
- Though, SpaceX is inferred to initiate these launches under strict conditions, where highly variant practices are unheard of
  - Will future data mirror test data conditions?
    - May need Logistic Regression 2 for highest precision, though it may not be ready for future variance
  - Expecting varied future scenarios?
    - Decision Tree 2 might be ideal as it is consistent and will anticipate variations in deployment data.

Conclusion

- This business problem is too complex to use simple insights such as Launch Success Rate over time in concluding statements
  - EDA was designed with attaching the analyst to the problem at hand and giving consistent understandings
  - EDA was also meant to open doors for other analytic angles should other business problems sprout, or for future considerations for the current problem
- In a field where low risk is paramount, Decision Tree 2 Classifier is chosen by SpaceY as the best model for this task
- Decision Making Refining
  - SpaceY's data science team will need more feature importance analysi
  - The models could use more robust error analyses
  - Additional validation should be considered if feasible.


Thank You!