Executive Summary - Justin Ferrales

Project Proposal:

I have made quite a few changes since the project proposal given from phase 1. I have made the switch from pokemon go onto launch vehicle analysis. I decided to make the switch because there was such little data on Pokemon Go so I started researching other topics and launch vehicle analysis lines up with my interests as I am an aerospace major. The goals I hope to complete with this project is to see the reliability and the success of different rockets, while varying country, launch site, and production costs. Later in my work, I plan on looking into rocket reusability, e.g. SpaceX's boosters, and do some analysis to see if rocket reusability in the future can be done.

Data Sources:

I have two main sources as of right now, but I plan on scraping a SpaceX API later, or finding a Kaggle dataset on the topic. I scraped a webpage "https://nextspaceflight.com/rockets/". In it I was able to get rocket name, number of missions, successes, partial failures, failures, success streak, and success rate. To do this I used beautiful soup to search for tags to get the aforementioned rocket elements above. Additionally, I used a kaggle dataset "https://www.kaggle.com/code/isaienkov/space-missions-eda-time-series-anaysis/input", which I will be using to compare costs and get data about launch site placement and time of flight. The observational unit in the first one is a singular rocket. The observational unit in the second one is a launch mission.

Main Variables of Interest

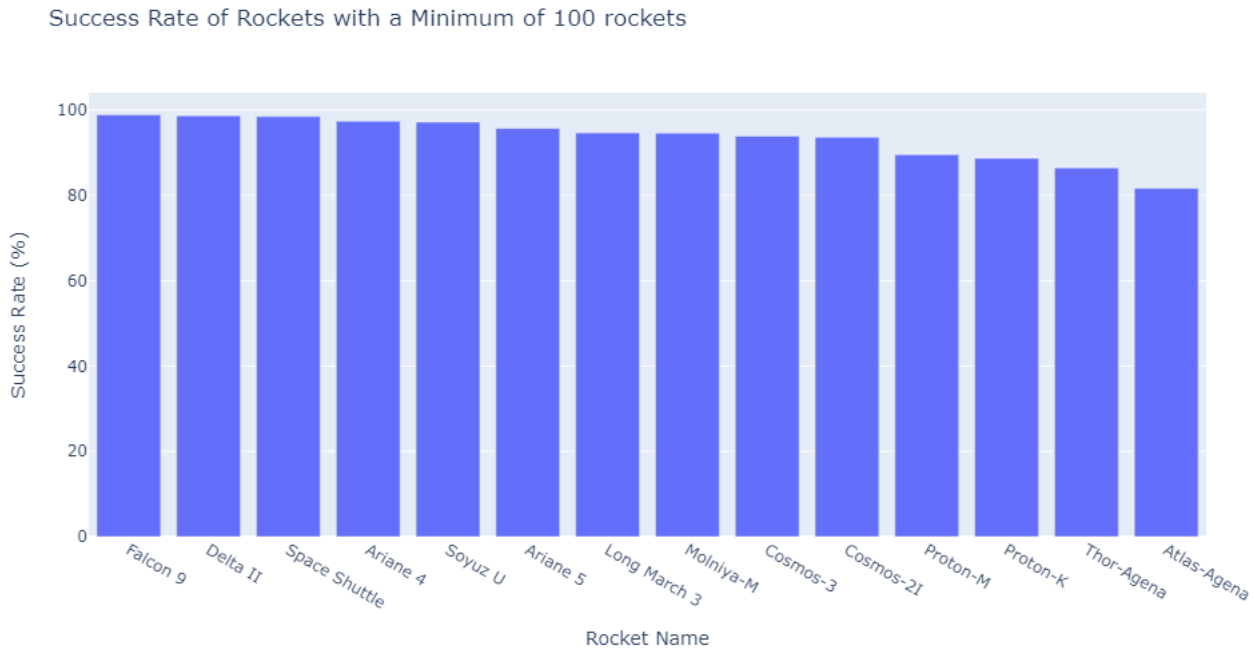
I plan on using all of the columns stated above to do my analysis. The website I scraped from has data dating back to the first rocket launch, Sputnik, all the way up to current times. They even have rocket launches planned, even one close to Cal Poly, as SpaceX will launch out of Vanderberg on Saturday 11/11.In my second dataset, there was such littler variables and I did additional research to get more values for machine learning later on.

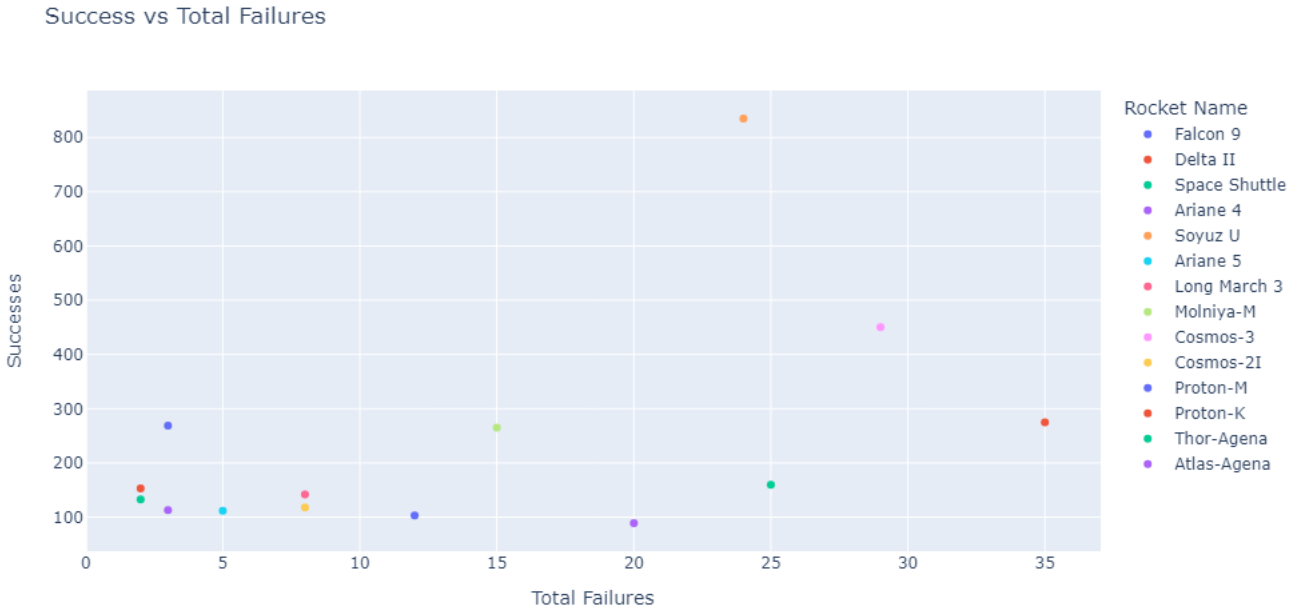
Data Cleaning/Processsing

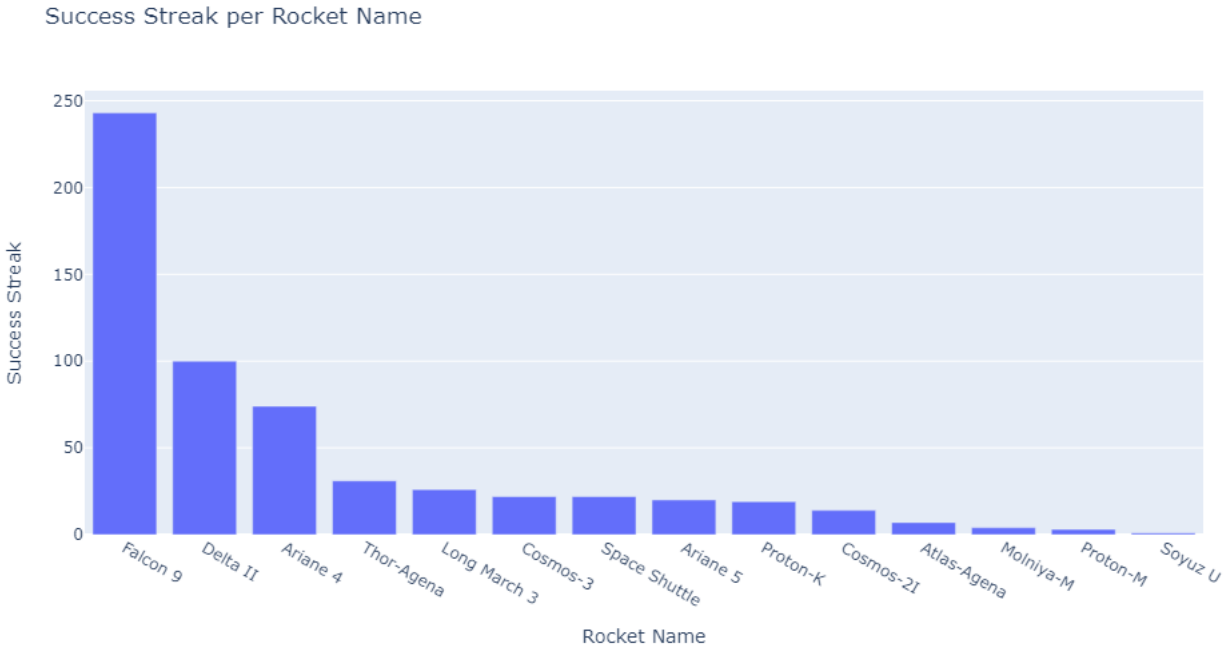
On my website scraping, I found that they used total failure and total success as a metric to compute success rate. However, I made a new column to make the total and partial failure, failure in total, and used the success to compute an accurate success rate.

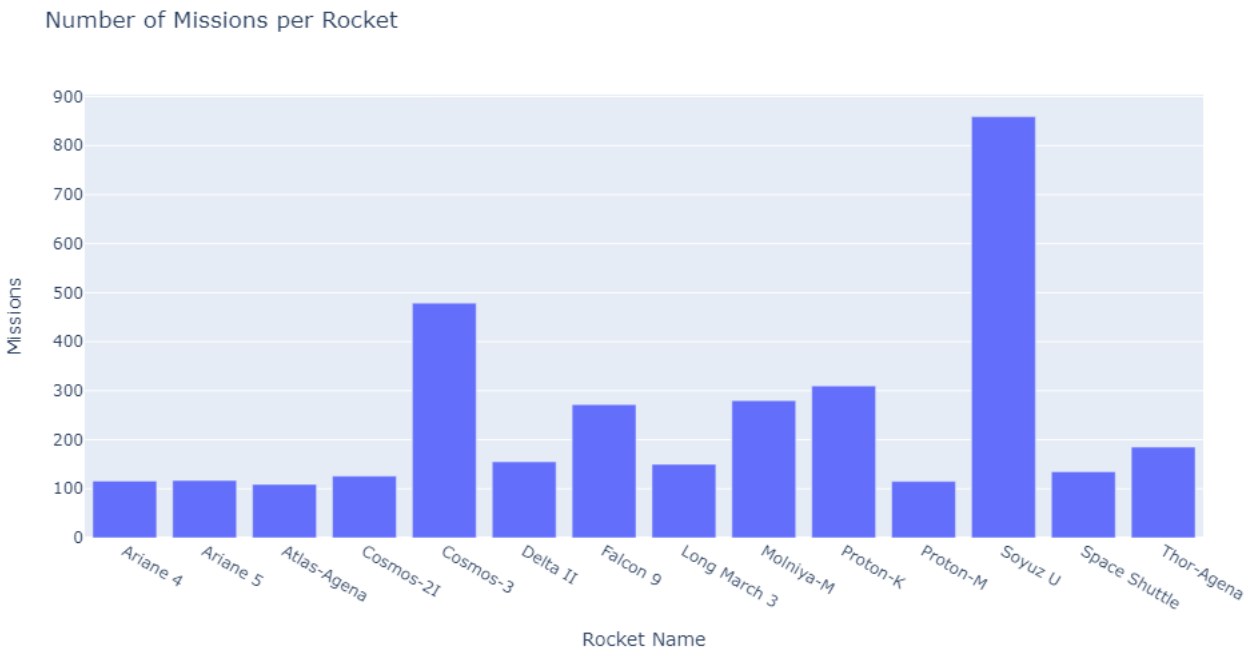
Success Rate = # of Successes / (# of Partial Failure + # of Failure)

In terms of data cleaning, I filled in all of the zeros in both the dataframes I am using. I am going to go down the list of what I have done so far. From the web scraping portion, I made several data frames for the various web pages, and I use pd.concat to concatenate all of the dataframes together. For both datasets, all of the data were casted as objects, so to do quantitative analysis on the columns I type casted them into integers. I had a lot of data on rockets, so I chose ones with high TRL (Tehcnology Readiness Level) and used number of launches as a TRL metric. I only used rockets with over 100 missions, for the first dataset.




Success vs Total Failures
Successes
Total Failures
0
5
10
15
20
25
30
35
100
200
300
400
500
600
700
800
Rocket Name
Falcon 9
Delta II
Space Shuttle
Ariane 4
Soyuz U
Ariane 5
Long March 3
Molniya-M
Cosmos-3
Cosmos-2I
Proton-M
Proton-K
Thor-Agena
Atlas-Agena


Success Streak per Rocket Name
Success Streak
Rocket Name
0
50
100
150
200
250
Falcon 9
Delta II
Ariane 4
Thor-Agena
Long March 3
Cosmos-3
Space Shuttle
Ariane 5
Proton-K
Cosmos-2I
Atlas-Agena
Molniya-M
Proton-M
Soyuz U



For the next dataset, I made various new columns to compute different things down the line. I wanted to see how the US performed against the world so I did some mapping like how we mapped passengers by class in the titanic data. I did that quite a bit, and here are some of the columns I came up with.

def values(c): val = c.strip() if 'USA' in c: return "USA" else: return "World" space["US_or_Not"] = space["Location"].map(values)

def successCount(c): if "Success" in c: return 1 else: return 0

space["binary_success"] = space["Status Mission"].map(successCount)

def launchPad(c): val = c.strip() p = val.split(',') return p[0]

space["Launch Pad"] = space["Location"].map(launchPad)

def launchCenter(c): val = c.strip() p = val.split(',') return p[1]

space["Launch Center"] = space["Location"].map(launchCenter)

def country(c): val = c.strip() p = val.split(',') if p[-1] == " New Mexico": return "USA" return p[-1]

space["Country"] = space["Location"].map(country)

def stateUS(c): val = c.strip() if "USA" in c: p = val.split(",") return p[2].strip() else: pass

space["State"] = space["Location"].map(stateUS)

def vehicle(c): val = c.split("|") return val[0]

space["Vehicle"] = space["Detail"].map(vehicle)

def DayofWeek(c): val = c.split() return val[0]

space["Day of the Week"] = space["Datum"].map(DayofWeek)

def month(c): val = c.split() return val[1]

space["Month"] = space["Datum"].map(month)

def day(c): val = c.split() new_string = val[2].replace(",", "") return new_string

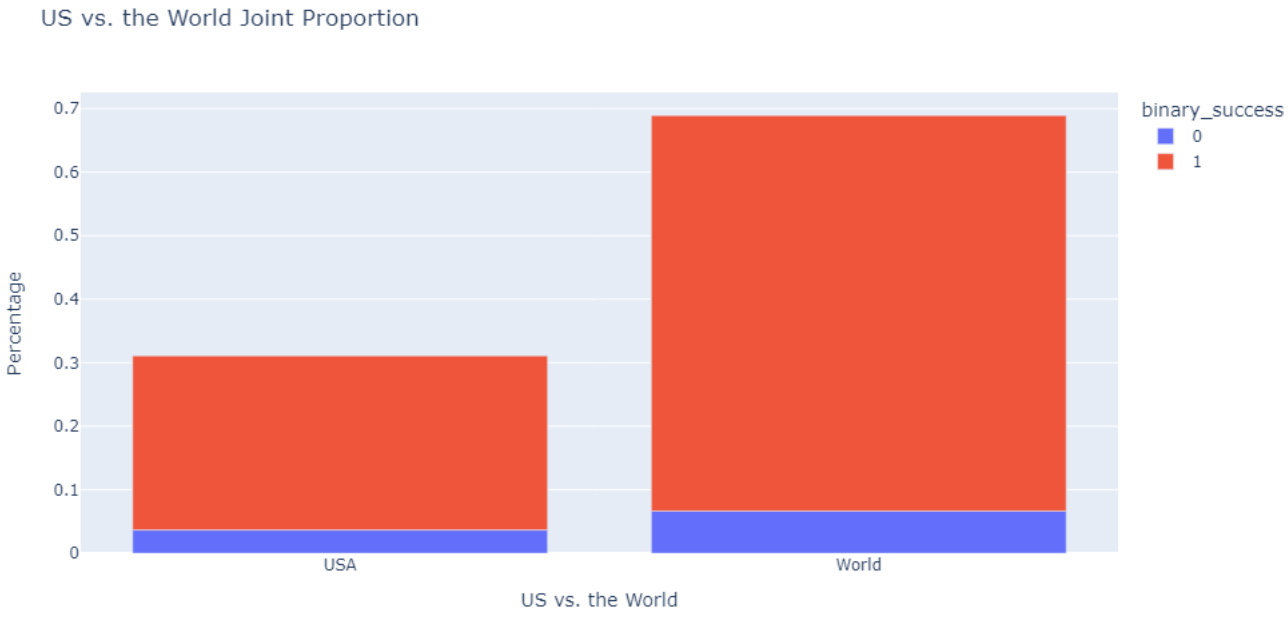
space["Day"] = space["Datum"].map(day)

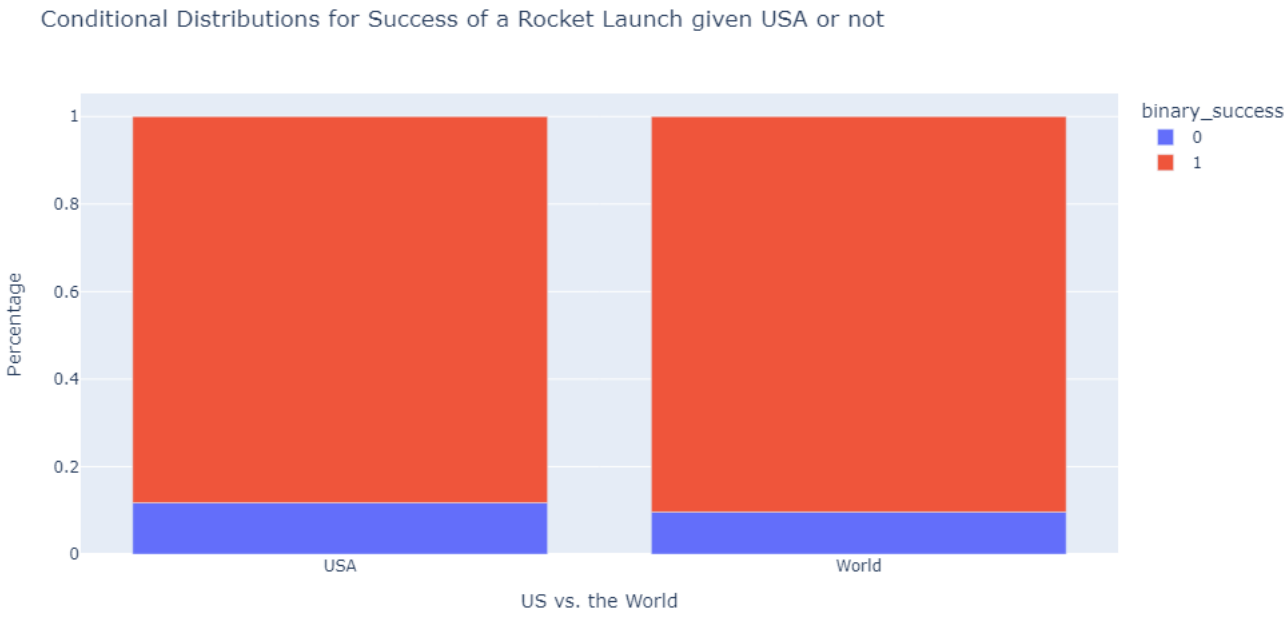
def type_agency(c): gov = ["CASIC", "Khrunichev", "CASC", "Roscosmos","JAXA", "VKS RF", "ISRO", "KARI","RVSN USSR","AMBA", "ESA", "NASA", "AEB", "US Air Force", "CNES", "RAE", "Armée de l'Air", "US Navy"] if c in gov: return "Government" else: return "Commercial"

space["Agency Type"] = space["Company Name"].map(type_agency)

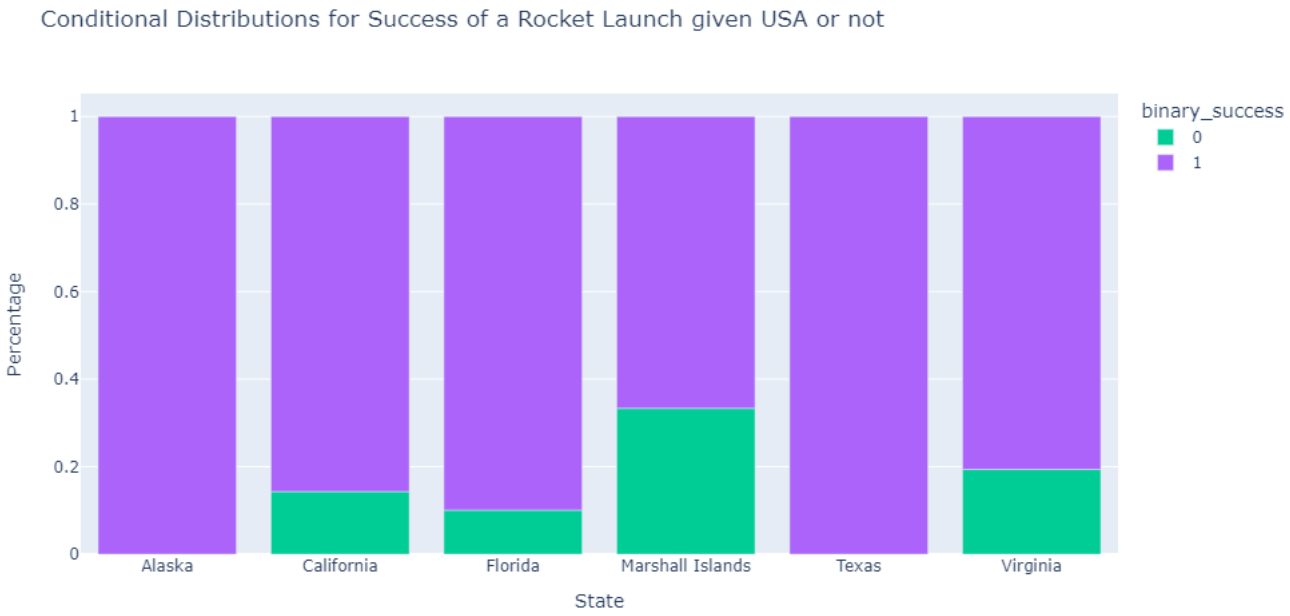
def numSatellites(c): count = 1 val = c.split("|") for item in val[1]: if item == "&": count += 1 return count space["Satellite Count"] = space["Detail"].map(numSatellites)

After that I computed some analysis using crosstab and groupby to compute the success of the US rockets. From the plots the US contributed 27.4% of total successes in the total history of rocketry. In terms of success rate the US has an 88% success rate which is 2% lower than the world's success rate at 90%.

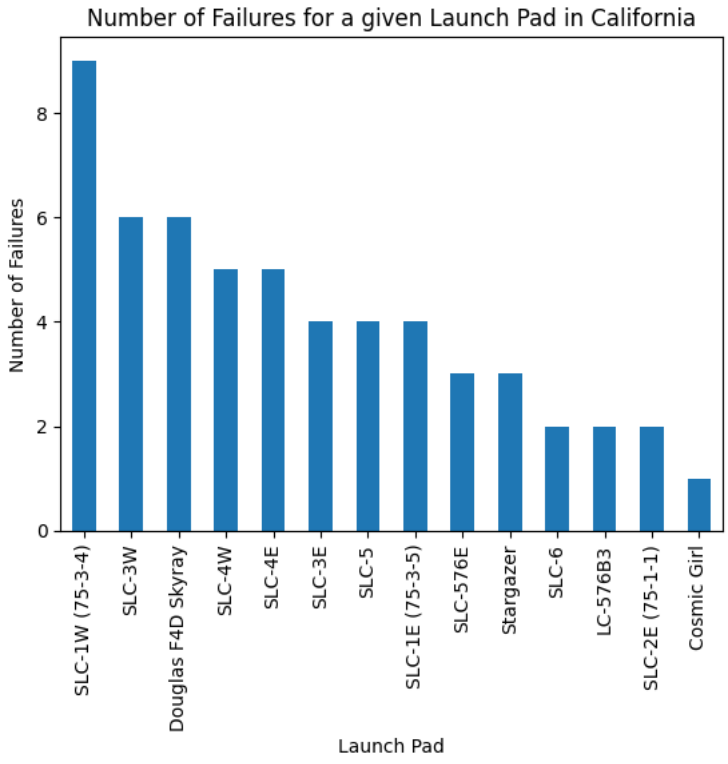

US vs. the World Joint Proportion
Percentage
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
USA
World
US vs. the World
binary_success
0
1

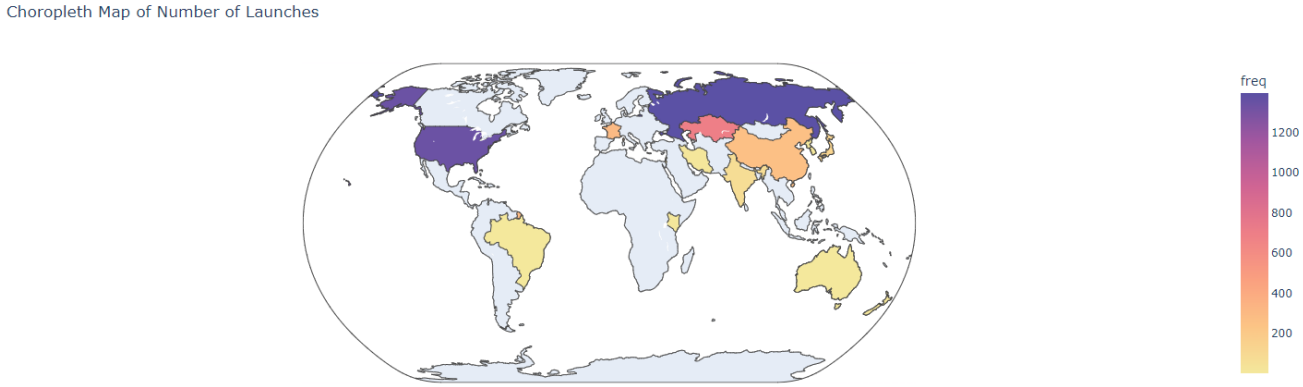

Conditional Distributions for Success of a Rocket Launch given USA or not
Percentage
1
0.8
0.6
0.4
0.2
0
USA
World
US vs. the World
binary_success
0
1

Keeping with the theme of US success, I wanted to see how the different launch sites affected reliability.



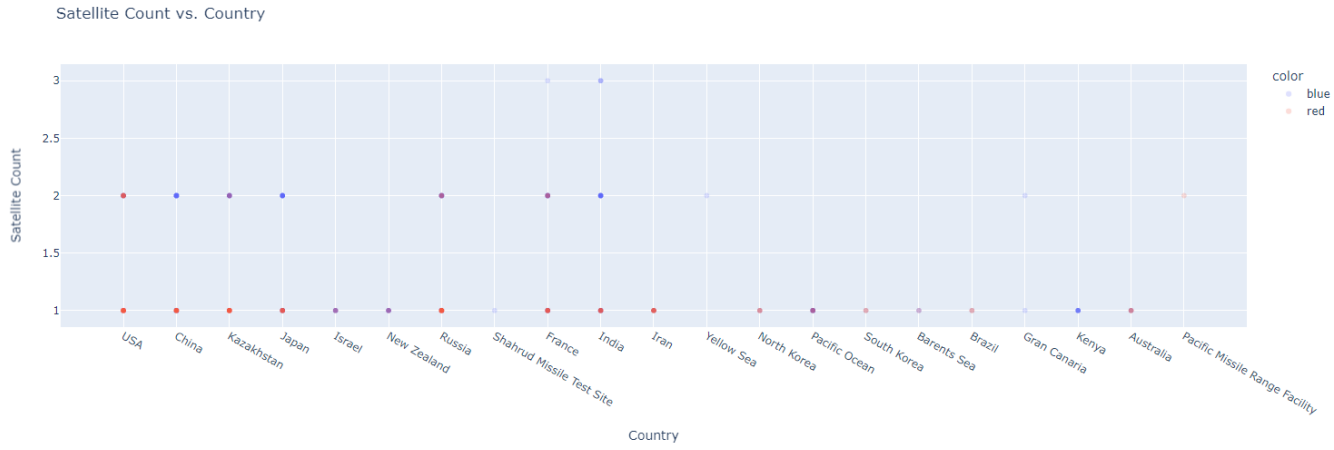
This data can be taken out of context as Alaska only had a few launches relative to the other states. CAlifornia and Florida are space hubs so their success rate is the bulk of the analysis. With that being said, I looked at the different launch pads in California, primarily here in Vanderberg Space Force Site (SLC).

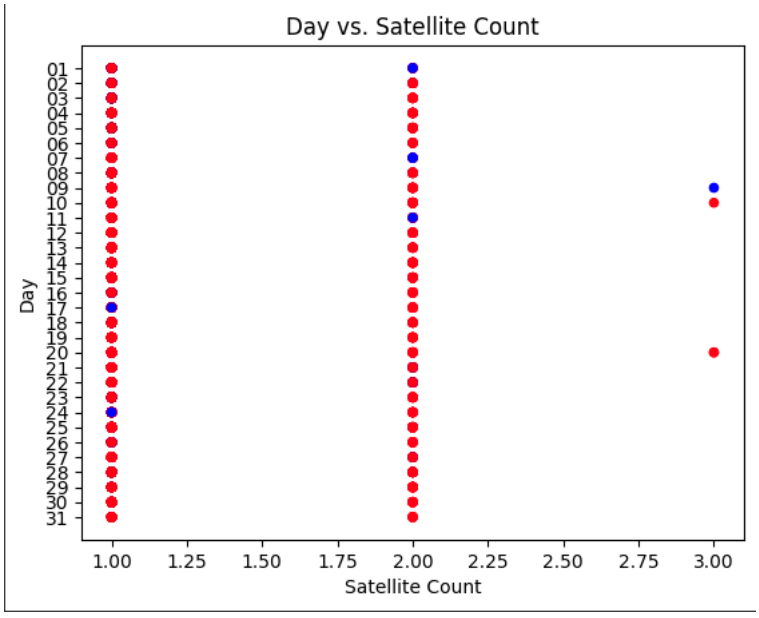


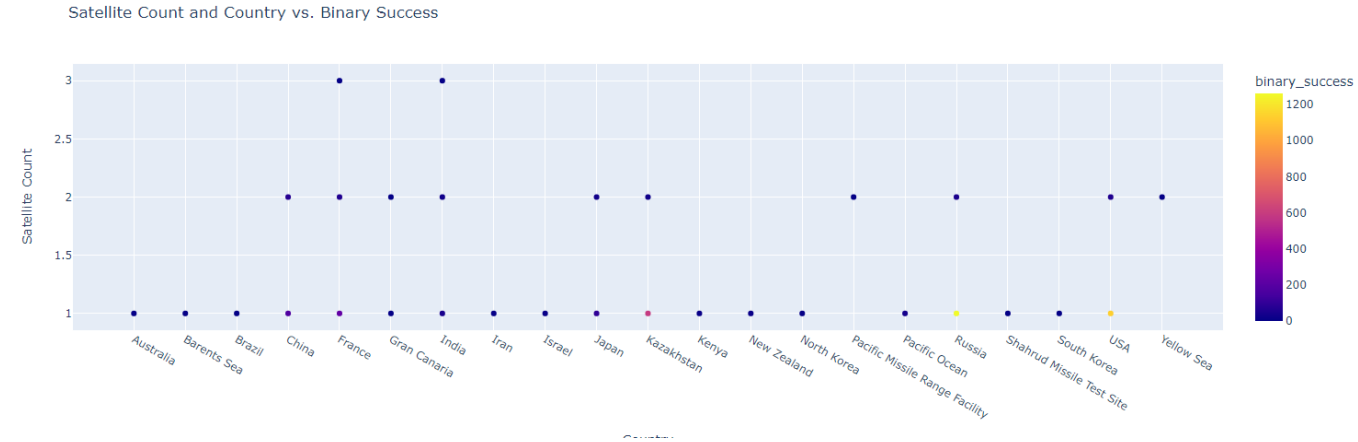


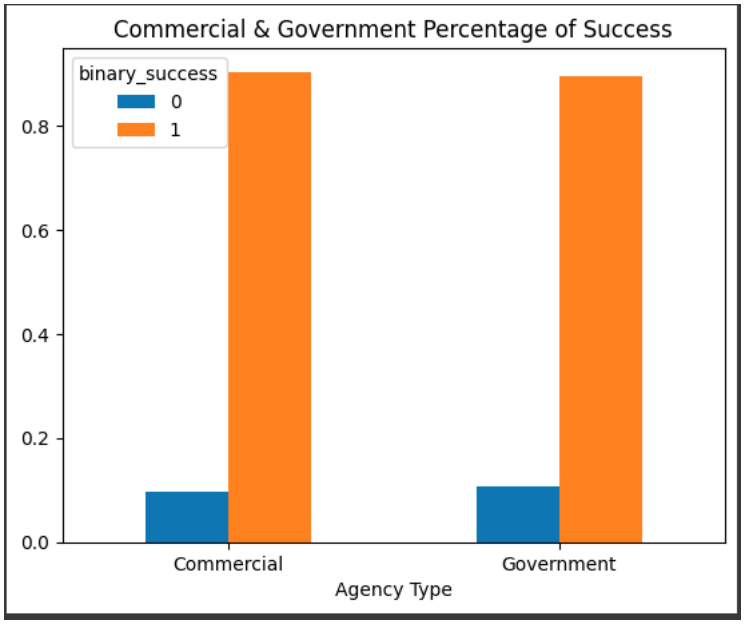
Part 3 Machine Learning

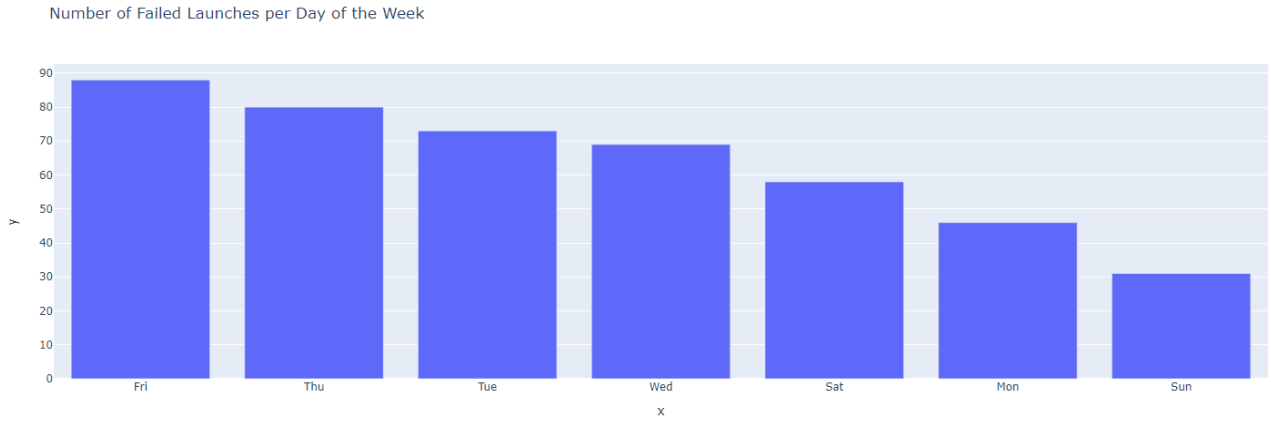
I wanted to find out what features could possibly lead me to building the best model possible.

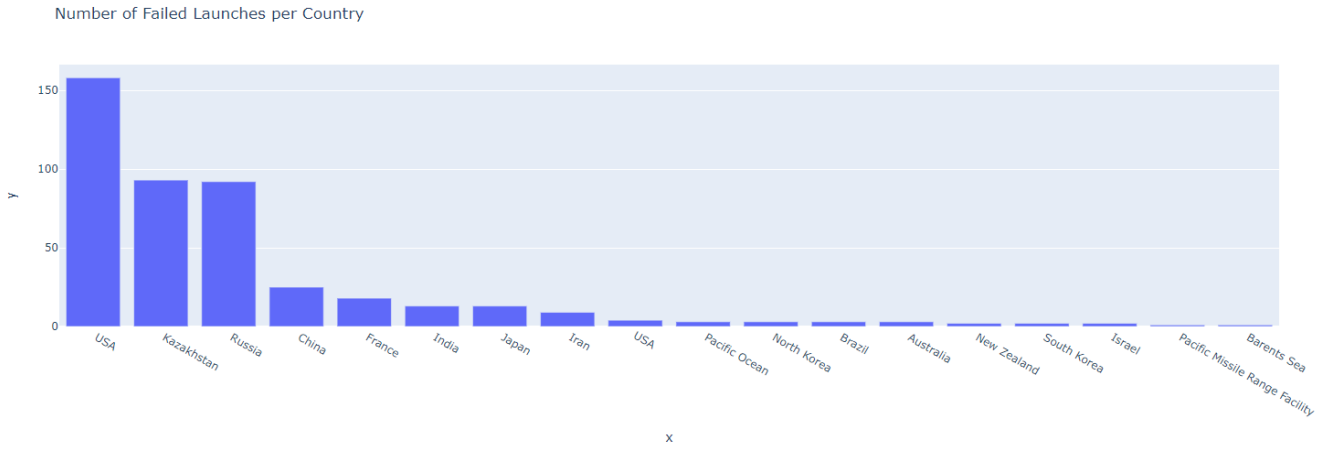





Satellite Count and Country vs. Binary Success
Satellite Count
binary_success
Australia
Barents Sea
Brazil
China
France
Gran Canaria
India
Iran
Israel
Japan
Kazakhstan
Kenya
New Zealand
North Korea
Pacific Missile Range Facility
Pacific Ocean
Russia
Shahrud Missile Test Site
South Korea
USA
Yellow Sea


Commercial & Government Percentage of Success
binary_success
0
1
Commercial
Government
Agency Type


Number of Failed Launches per Day of the Week
Fri
Thu
Tue
Wed
Sat
Mon
Sun
x
y



Description of the model

I started off by getting my data that did not have any NaN's namely in the Cost column of my dataframe. Following this I ended up using KNeighborsClassifier and using the columns: Day, Satellite Count, Cost, Agency Type, Day of the Week, and the thing I wanted to test was the success of the mission. For the categorical values I used a one hot encoder and defined a pipeline, using KNeighborsClassifier. I also did a seperate model predicting the cost of a rocket per its satellite count, country of origin, month, and agency type, i.e. commericial or government entity.

This is for the first model described

from sklearn.preprocessing import StandardScaler from sklearn.neighbors import KNeighborsClassifier from sklearn.pipeline import make_pipeline

from sklearn.compose import make_column_transformer from sklearn.preprocessing import OneHotEncoder

x_train = df_ml[["Day", "Satellite Count", "Cost", "Agency Type", "Day of the Week"]] x_test = df_test[["Day","Agency Type", "Day of the Week"]] y_train = df_ml["Status Mission"]

ct = make_column_transformer((OneHotEncoder(), ["Agency Type", "Day of the Week"]), remainder="drop" # all other columns in X will be dropped.)

pipeline = make_pipeline(ct, KNeighborsClassifier(n_neighbors=5))

pipeline.fit(x_train, y_train)

Second Model

ct = make_column_transformer((StandardScaler(), ["Satellite Count"]), (OneHotEncoder(), ["Country", "Month", "Agency Type"]), remainder = "drop")

from sklearn.pipeline import make_pipeline from sklearn.neighbors import KNeighborsRegressor pipeline1 = make_pipeline(ct, KNeighborsRegressor(n_neighbors=5))

pipeline1.fit(X=df_ml[["Satellite Count","Country", "Month", "Agency Type"]], y=df_ml["Cost"])

Description of new models

I realized that my preliminary model only predicted successful launches because I was using a filtered dataframe that solely had successful launches. I then changed it so that the training data is on the entire dataframe. I then followed the same approach being a KNeighborsClassification and choosing some of the features from the graphs above. Additionally, I had to use a column trasnformer to deal with the categorical variables.

1st model

```
import pandas as pd
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.neighbors import KNeighborsClassifier
from sklearn.pipeline import make_pipeline
from sklearn.compose import make_column_transformer

ct = make_column_transformer(
    (OneHotEncoder(handle_unknown='ignore'), ["Month", "Country", "Agency Type"]), remainder = "passthrough"
)

# this is my training data

xTT = space[["Month", "Country", "Agency Type", "Satellite Count"]]
yTT = space["binary_success"]

# define a pipeline
pipeline20 = make_pipeline(
    ct,
    KNeighborsClassifier(n_neighbors=2)
)

pipeline20.fit(xTT, yTT)
a = pipeline20.predict(xTT)
pd.Series(a).value_counts()
```

```
1    3815
0     509
dtype: int64
```

2nd model

```
ct = make_column_transformer(
    (OneHotEncoder(), ["Month", "Day of the Week", "Agency Type"]), remainder = "passthrough"
)

# this is my training data

xT1 = space[["Month", "Day of the Week", "Agency Type", "Satellite Count"]]
yT1 = space["binary_success"]
pipeline0 = make_pipeline(
    ct,
    KNeighborsClassifier(n_neighbors=2)
)

pipeline0.fit(xT1, yT1)
```

```
Pipeline
```

3rd Model

```
ct_new = make_column_transformer(
    (OneHotEncoder(), ["Month", "Day of the Week"]), remainder = "passthrough"
)

# this is my training data

xT_new = space[["Month", "Day of the Week"]]
yT_new = space["binary_success_label"]

# define a pipeline
pipeline_new = make_pipeline(
    ct_new,
    KNeighborsClassifier(n_neighbors=2)
)

pipeline_new.fit(xT_new, yT_new)
```

I also performed grid search to find the best k to use for the model.

Double-click (or enter) to edit

Results

Overall, my results were saddening. For my first model, it always predicted a successful launch which is not the case. In the dataframe there were 400 failures, so there should be some percentage of failure. I think it is due to the variability of the reasons why a rocket failed. Trying to predict whether or not a rocket failed based on day, number of satellites, cost, agency type, and day of the week are not appropriate metrics to

determine the success of a rocket. Additionally, since there is such an overwhelming majority of successes by using K nearest neighbors, the model will look at the nearest neighbors and always predict a successful launch. Even when plugging in exact values for a failed launch, due to the nature of K nearest neighbors it predicts a successful launch which is disheartning.

New Results

For the three models I was able to get predictions for failures and successes which is a sharp improvement from before. The reason I was able to predict failures and success this time was because I used the entire data frame instead of a filter data frame. The resulting cv mean for the three models were: 0.812, 0.771, 0.796. As we can see from the cv scores the best model is the first one. However, the f1score, precision, and recall for predicting success is significantly higher than predicting failure. This could be due to bad features chosen in the model, as well as a bad data set. There are almost 4,000 successes to 400 failures in a data set, thus the model will predict successes at a higher rate. For my third model, I also had a f1 score, recall, precision score of 100% for both failure and success. This is definitely not possible, and I think that this attributed to overfitting, as this is considered a perfect model. I think that since the features chosen "month", "day of the week" are pretty unique it could have led to this result. Additionally, it is hard to tell if a rocket fail due to the specified features because a rocket can fail for a multitude of reasons, some being due to improper testing and faulty structures.

```
!wget -nc https://raw.githubusercontent.com/brpy/colab-pdf/master/colab_pdf.py
from colab_pdf import colab_pdf
colab_pdf('Executive Summary.ipynb')
```

```
File ‘colab_pdf.py’ already there; not retrieving.


WARNING: apt does not have a stable CLI interface. Use with caution in scripts.


WARNING: apt does not have a stable CLI interface. Use with caution in scripts.

E: Unable to locate package texlive-generic-recommended
[NbConvertApp] WARNING | pattern '$notebookpath$file_name' matched no files
This application is used to convert notebook files (*.ipynb)
        to various other formats.

        WARNING: THE COMMANDLINE INTERFACE MAY CHANGE IN FUTURE RELEASES.

Options
=======
The options below are convenience aliases to configurable class-options,
as listed in the "Equivalent to" description-line of the aliases.
To see all configurable class-options for some <cmd>, use:
    <cmd> --help-all

--debug
    set log level to logging.DEBUG (maximize logging output)
    Equivalent to: [--Application.log_level=10]
--show-config
    Show the application's configuration (human-readable format)
    Equivalent to: [--Application.show_config=True]
--show-config-json
    Show the application's configuration (json format)
    Equivalent to: [--Application.show_config_json=True]
--generate-config
    generate default config file
    Equivalent to: [--JupyterApp.generate_config=True]
-y
    Answer yes to any questions instead of prompting.
    Equivalent to: [--JupyterApp.answer_yes=True]
--execute
    Execute the notebook prior to export.
    Equivalent to: [--ExecutePreprocessor.enabled=True]
--allow-errors
    Continue notebook execution even if one of the cells throws an error and include the error message in the cell output (the defau
    Equivalent to: [--ExecutePreprocessor.allow_errors=True]
--stdin
    read a single notebook file from stdin. Write the resulting notebook with default basename 'notebook.*'
    Equivalent to: [--NbConvertApp.from_stdin=True]
--stdout
    Write notebook output to stdout instead of files.
    Equivalent to: [--NbConvertApp.writer_class=StdoutWriter]
--inplace
    Run nbconvert in place, overwriting the existing notebook (only
            relevant when converting to notebook format)
    Equivalent to: [--NbConvertApp.use_output_suffix=False --NbConvertApp.export_format=notebook --FilesWriter.build_directory=]
--clear-output
    Clear output of current file and save in place,
            overwriting the existing notebook.
    Equivalent to: [--NbConvertApp.use_output_suffix=False --NbConvertApp.export_format=notebook --FilesWriter.build_directory= --Cl
--no-prompt
    Exclude input and output prompts from converted document.
    Equivalent to: [--TemplateExporter.exclude_input_prompt=True --TemplateExporter.exclude_output_prompt=True]
--no-input
    Exclude input cells and output prompts from converted document.
            This mode is ideal for generating code-free reports.
    Equivalent to: [--TemplateExporter.exclude_output_prompt=True --TemplateExporter.exclude_input=True --TemplateExporter.exclude_i
--allow-chromium-download
    Whether to allow downloading chromium if no suitable version is found on the system.
    Equivalent to: [--WebPDFExporter.allow_chromium_download=True]
```