jferrales_analysisandml

December 11, 2023

[240]:
```python
import pandas as pd
from bs4 import BeautifulSoup
import requests
```

Grabbing all the websites that have to do with my rocket data acquisition.

[241]:
```python
response = requests.get("https://nextspaceflight.com/rockets/")
response1 = requests.get("https://nextspaceflight.com/rockets/?page=1&search=")
response2 = requests.get("https://nextspaceflight.com/rockets/?page=2&search=")
response3 = requests.get("https://nextspaceflight.com/rockets/?page=3&search=")
response4 = requests.get("https://nextspaceflight.com/rockets/?page=4&search=")
response5 = requests.get("https://nextspaceflight.com/rockets/?page=5&search=")
response6 = requests.get("https://nextspaceflight.com/rockets/?page=6&search=")
response7 = requests.get("https://nextspaceflight.com/rockets/?page=&search=")
```

Parsing the html using BeautifulSoup

[242]:
```python
soup = BeautifulSoup(response.content, "html.parser")
soup1 = BeautifulSoup(response1.content, "html.parser")
soup2 = BeautifulSoup(response2.content, "html.parser")
soup3 = BeautifulSoup(response3.content, "html.parser")
soup4 = BeautifulSoup(response4.content, "html.parser")
soup5 = BeautifulSoup(response5.content, "html.parser")
soup6 = BeautifulSoup(response6.content, "html.parser")
soup7 = BeautifulSoup(response7.content, "html.parser")
```

Function for parsing my 8 soups.

[243]:
```python
def getRocketData(soup):
  rocket_sections = soup.find_all('section', class_='card section--center␣
↪mdl-grid mdl-grid--no-spacing mdl-shadow--6dp')

  rocket_data = []

  for section in rocket_sections:
      # Extract rocket name
      rocket_name = section.find('h5').text
```

```
    info_elements = section.find_all('div', class_='mdl-cell␣
↪mdl-cell--12-col-desktop mdl-cell--12-col-tablet')
    #rocket_info = {element.text.split(': ')[0]: element.text.split(': ')[1]␣
↪for element in info_elements}
    rocket_info = {}
    for element in info_elements:
      key, value = element.text.split(': ')
      rocket_info[key] = value
    # Append data to the list, this unpacks the dictionary I built
    rocket_data.append({'Rocket Name': rocket_name, **rocket_info})

  return (rocket_data)
```

[244]:
```
a = getRocketData(soup)
b = getRocketData(soup1)
c = getRocketData(soup2)
d = getRocketData(soup3)
e = getRocketData(soup4)
f = getRocketData(soup5)
g = getRocketData(soup6)
h = getRocketData(soup7)
```

[245]:
```
dfp1 = pd.DataFrame(a)
dfp2 = pd.DataFrame(b)
dfp3 = pd.DataFrame(c)
dfp4 = pd.DataFrame(d)
dfp5 = pd.DataFrame(e)
dfp6 = pd.DataFrame(f)
dfp7 = pd.DataFrame(g)
```

I accidently made dfp1 the same as dfp2 so I just took it off.

[246]:
```
df_rockets = pd.concat([dfp2, dfp3, dfp4, dfp5, dfp6, dfp7], ignore_index =␣
↪True)
```

Final dataframe for now thank god

Gonna do some cleaning. I am going to take out all of the NaN's and I do not understand the given success rate. Successes is 1 and failure is 1 and partial failure is 1. I am going to treat each one as a failure entirely.

[247]:
```
df_rockets.fillna(0, inplace=True)
```

[248]:
```
df_rockets.head()
```

[248]:
```
  Rocket Name Missions Successes Partial Failures Failures Success Streak  \
0       Alpha        3         1                1        1              1
1        Amur        0         0                0        0              0
```

```
2     Angara 1         3          3                 0        0           3
3   Angara A5          3          2                 0        1           0
4     Antares         18         17                 0        1          13

  Success Rate
0        50.0%
1            0
2         100%
3        66.7%
4        94.4%
```

Some more cleaning. The Successes and failures were casted in as objects so now I am casting them as integers to do some analysis on them further.

[249]:
```
df_rockets.Successes = df_rockets.Successes.astype(int)
df_rockets.Failures = df_rockets.Failures.astype(int)
df_rockets["Partial Failures"] = df_rockets["Partial Failures"].astype(int)
df_rockets.Missions = df_rockets.Missions.astype(int)
df_rockets["Success Streak"] = df_rockets["Success Streak"].astype(int)
```

[250]:
```
df_rockets["Accurate Success Rate"] = ((df_rockets["Successes"] /␣
 ↪(df_rockets["Missions"])) * 100)
df_rockets["Failure Rate"] = (df_rockets["Failures"] + df_rockets["Partial␣
 ↪Failures"]) / df_rockets.Missions
df_rockets["Total Failures"] = df_rockets.Failures + df_rockets["Partial␣
 ↪Failures"]
df_rockets.fillna(0, inplace=True)
```

Check the most reliable rockets, upon research apparently plotly plots stuff by how they appear in the dataframe, I wanted success rate to be in order by descending so I did so.

From early inspection it seems that these 100% range for 1 for 1, I should check for higher count of missions. I decided that rockets with high TRL (technological readiness level) should have at least 100 flights. I realize that this results in a small dataset, but I just want to see what rockets are successful and put them into a visualization.

[251]:
```
df_sorted = df_rockets.sort_values(by='Accurate Success Rate', ascending =␣
 ↪False)
df_TRL = df_rockets[df_rockets.Missions > 100]
df_TRLsorted = df_TRL.sort_values(by=['Accurate Success Rate', 'Missions',␣
 ↪'Success Streak'], ascending = False)
df_TRL_success = df_TRL.sort_values(by = "Success Streak", ascending = False)
df_TRL
```

[251]:
```
      Rocket Name  Missions  Successes  Partial Failures  Failures  \
8        Ariane 4       116        113                 0         3
9        Ariane 5       117        112                 3         2
```

```
15     Atlas-Agena     109      89          5     15
37       Cosmos-2I     126     118          0      8
38        Cosmos-3     479     450          8     21
52        Delta II     155     153          1      1
67        Falcon 9     282     279          1      2
98    Long March 3     150     142          6      2
116      Molniya-M     280     265         12      3
133       Proton-K     310     275          2     33
134       Proton-M     115     103          4      8
164        Soyuz U     859     835          2     22
165  Space Shuttle     135     133          0      2

     Success Streak Success Rate  Accurate Success Rate  Failure Rate  \
8                74        97.4%              97.413793      0.025862
9                20        97.0%              95.726496      0.042735
15                7        83.9%              81.651376      0.183486
37               14        93.7%              93.650794      0.063492
38               22        94.8%              93.945720      0.060543
52              100        99.0%              98.709677      0.012903
67              253        99.1%              98.936170      0.010638
98               26        96.7%              94.666667      0.053333
116               4        96.8%              94.642857      0.053571
133              19        89.0%              88.709677      0.112903
134               3        91.3%              89.565217      0.104348
164               1        97.3%              97.206054      0.027939
165              22        98.5%              98.518519      0.014815

     Total Failures
8                 3
9                 5
15               20
37                8
38               29
52                2
67                3
98                8
116              15
133              35
134              12
164              24
165               2
```

Using plotly to create a bar chart of the success rate of rockets with a minumum of 100 launches.

[252]:
```
import plotly.express as px
```

```
px.bar(df_TRLsorted, x="Rocket Name", y="Accurate Success Rate", title="Success␣
↪Rate of Rockets with a Minimum of 100 launches",
       labels = {"Rocket Name": "Rocket Name", "Accurate Success Rate":␣
↪"Success Rate (%)"})
```

Note: This format came from plotly I used it as a reference to make my points larger.

[253]:
```
#df = px.data.iris()
#fig = px.scatter(df, x="sepal_width", y="sepal_length", color="species")

#fig.update_traces(marker=dict(size=12,
                              #line=dict(width=2,
                               #         color='DarkSlateGrey')),
                  #selector=dict(mode='markers'))
#fig.show()
```

[254]:
```
fig = px.scatter(df_TRLsorted, x="Total Failures", y = "Successes", color =␣
↪"Rocket Name", title="Success vs Total Failures for Rockets with over 100␣
↪Launches")
fig.update_traces(marker=dict(size=8))
fig.update_layout(hovermode='closest')
fig.show()
```

From the plot below we see that most launch vehicles do not go over 200 launches. The success rate of the launches are clustered up in between 100 launches or so.

[255]:
```
px.scatter(df_rockets, x="Missions", y= "Accurate Success Rate", color="Rocket␣
↪Name", title="Overall Success Rate (%) for all Rockets")
```

Success Streak of Rockets

[256]:
```
px.bar(df_TRL_success, x="Rocket Name", y = "Success Streak", title="Success␣
↪Streak per Rocket Name")
```

Total number of missions per rocket

[257]:
```
px.bar(df_rockets, x = "Rocket Name", y = "Missions", title="Number of Missions␣
↪per Rocket")
```

[258]:
```
fig = px.pie(df_TRL_success, values ="Missions", names="Rocket Name",␣
↪title="Proportion of Launches per Rocket Relative to Other Rockets with High␣
↪TRL ")
fig.update_traces(textposition="inside", textinfo = 'percent+label')
```

1 New Section: Using Kaggle Dataset for mission launches.

[259]:
```
space = pd.read_csv("Space_Corrected.csv")
space.head()
```

[259]:
```
   Unnamed: 0.1  Unnamed: 0 Company Name  \
0             0           0       SpaceX
1             1           1         CASC
2             2           2       SpaceX
3             3           3    Roscosmos
4             4           4          ULA

                                            Location  \
0         LC-39A, Kennedy Space Center, Florida, USA
1  Site 9401 (SLS-2), Jiuquan Satellite Launch Ce…
2                      Pad A, Boca Chica, Texas, USA
3       Site 200/39, Baikonur Cosmodrome, Kazakhstan
4           SLC-41, Cape Canaveral AFS, Florida, USA

                       Datum                                        Detail  \
0  Fri Aug 07, 2020 05:12 UTC  Falcon 9 Block 5 | Starlink V1 L9 & BlackSky
1  Thu Aug 06, 2020 04:01 UTC           Long March 2D | Gaofen-9 04 & Q-SAT
2  Tue Aug 04, 2020 23:57 UTC            Starship Prototype | 150 Meter Hop
3  Thu Jul 30, 2020 21:25 UTC  Proton-M/Briz-M | Ekspress-80 & Ekspress-103
4  Thu Jul 30, 2020 11:50 UTC                    Atlas V 541 | Perseverance

  Status Rocket  Rocket Status Mission
0  StatusActive   50.0         Success
1  StatusActive  29.75         Success
2  StatusActive     NaN        Success
3  StatusActive   65.0         Success
4  StatusActive  145.0         Success
```

*checks to see if my code works

[260]:
```
f = space.loc[0]
#string = f.Location.split(',')
val = f.Location.strip()
pp = val.split(',')


location_list = ['LC-39A', 'Kennedy Space Center', 'Florida', 'USA']

if 'USA' in val:
    print("USA is in the list.")
else:
    print("USA is not in the list.")
```

```
print(location_list)
print(val)
```

```
USA is in the list.
['LC-39A', 'Kennedy Space Center', 'Florida', 'USA']
LC-39A, Kennedy Space Center, Florida, USA
```

Mapping values like how we did in the titanic notebooks. I wanted to see if a given mission was USA sponsored or not, and I wanted to change Status from StatusActive to just Active. Just felt like it was smoother, cleaner, easier to read that way.

[261]:
```
def values(c):
  val = c.strip()
  if 'USA' in c:
    return "USA"
  else:
    return "World"


def Status(c):
  if c == "StatusActive":
    return "Active"
  else:
    return "Inactive"

space["Status"] = space["Status Rocket"].map(Status)
space["US_or_Not"] = space["Location"].map(values)
```

We have the same thing going on here like we did prior section. I am going to consider partial and prelaunch failure as failure as a whole.

[262]:
```
space["Status Mission"].value_counts()
```

[262]:
```
Success              3879
Failure               339
Partial Failure       102
Prelaunch Failure       4
Name: Status Mission, dtype: int64
```

Kind of like how we did it for the classes for malignant and benign. I wanted to turn the successes into something binary in terms of 0 & 1. I also wanted to do it for failure count because why not.

[263]:
```
def successCount(c):
  if "Success" in c:
    return 1
  else:
```

```
    return 0

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

def failureCount(c):
  if "Failure" in c:
    return 1
  else:
    return 0

space["failureCount"] = space["Status Mission"].map(failureCount)
```

[264]:
```
space.binary_success.value_counts()
```

[264]:
```
1    3879
0     445
Name: binary_success, dtype: int64
```

[265]:
```
space.failureCount.value_counts()
```

[265]:
```
0    3879
1     445
Name: failureCount, dtype: int64
```

Wanted to see how the USA fairs on a global scale vs the world in terms of rocket success and failure

[266]:
```
joint_success = pd.crosstab(
    space["US_or_Not"], space["binary_success"], normalize = True
)

# abels = {"Rocket Name": "Rocket Name", "Accurate Success Rate": "Success Rate␣
 ↪(%)"}



px.bar(joint_success, title="US vs. the World Joint Proportion",
       labels = {"US_or_Not": "US vs. the World", "value": "Percentage"})
```

[267]:
```
success_given_country = joint_success.divide(
    joint_success.sum(axis=1),axis=0
)
success_given_country
```

[267]:
```
binary_success         0         1
US_or_Not
USA             0.117560  0.882440
World           0.096309  0.903691
```

In more USA vs the world graphs, we see that the USA has a slightly lower percentage of success versus the world, and a higher percentage of failure. This is probably due to the rockets shown prior in my first stage of data collection. The soyuz/cosmos are reliable rockets used multiple times with high success rate, so that can overpower the USA's success especially with our early struggle in apollo missions and our commercial rocket endeavors.

[268]:
```
px.bar(success_given_country, labels = {"US_or_Not": "US vs. the World",␣
↪"value": "Percentage"},
       title = "Conditional Distributions for Success of a Rocket Launch given␣
↪USA or not")
```

More cleaning for some reason it was red in w/ question marks just replacing.

[269]:
```
value_to_replace = "Arm??e de l'Air"
replacement_value = "Armée de l'Air"
space['Company Name'] = space['Company Name'].replace(value_to_replace,␣
↪replacement_value)
space["Company Name"].unique()
```

[269]:
```
array(['SpaceX', 'CASC', 'Roscosmos', 'ULA', 'JAXA', 'Northrop', 'ExPace',
       'IAI', 'Rocket Lab', 'Virgin Orbit', 'VKS RF', 'MHI', 'IRGC',
       'Arianespace', 'ISA', 'Blue Origin', 'ISRO', 'Exos', 'ILS',
       'i-Space', 'OneSpace', 'Landspace', 'Eurockot', 'Land Launch',
       'CASIC', 'KCST', 'Sandia', 'Kosmotras', 'Khrunichev', 'Sea Launch',
       'KARI', 'ESA', 'NASA', 'Boeing', 'ISAS', 'SRC', 'MITT', 'Lockheed',
       'AEB', 'Starsem', 'RVSN USSR', 'EER', 'General Dynamics',
       'Martin Marietta', 'Yuzhmash', 'Douglas', 'ASI', 'US Air Force',
       'CNES', 'CECLES', 'RAE', 'UT', 'OKB-586', 'AMBA', "Armée de l'Air",
       'US Navy'], dtype=object)
```

2 Discussion on further parts of the project

After attempts at machine learning using the provided dataset from kaggle and feedback from Dr. Ross, I realized that I may not have enough features to apply machine learning to this project. With that being said, embedded within ceratin columns were various other parameters that I could map values to. Namely launch pad, launch center, country, and state (given that it is a USA country). Additionally, I was able to attain the launch vehicle that was used, and the number of satellites aboard each mission (although this may have been a guess in the website each one had a major satellite on board, but from prior aerospace classes we know that most launch vehicles take up mini satellites like the ones we have at Cal Poly such as Cubesats. But I am sure that they did not include those in the launch detail). I was also able to get the day of the launch (numerical), the actual day of the wekk, and the month. From the company names, i did some additional research to find out whether or not these companies were commercial or government.

[270]:
```python
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)
```

[271]:
```
space.Detail.unique()
```

[271]:
```
array(['Falcon 9 Block 5 | Starlink V1 L9 & BlackSky',
       'Long March 2D | Gaofen-9 04 & Q-SAT',
       'Starship Prototype | 150 Meter Hop', …,
       'Vanguard | Vanguard TV3', 'Sputnik 8K71PS | Sputnik-2',
       'Sputnik 8K71PS | Sputnik-1'], dtype=object)
```

[272]:
```
space["Company Name"].unique()
```

[272]:
```
array(['SpaceX', 'CASC', 'Roscosmos', 'ULA', 'JAXA', 'Northrop', 'ExPace',
       'IAI', 'Rocket Lab', 'Virgin Orbit', 'VKS RF', 'MHI', 'IRGC',
       'Arianespace', 'ISA', 'Blue Origin', 'ISRO', 'Exos', 'ILS',
       'i-Space', 'OneSpace', 'Landspace', 'Eurockot', 'Land Launch',
       'CASIC', 'KCST', 'Sandia', 'Kosmotras', 'Khrunichev', 'Sea Launch',
```

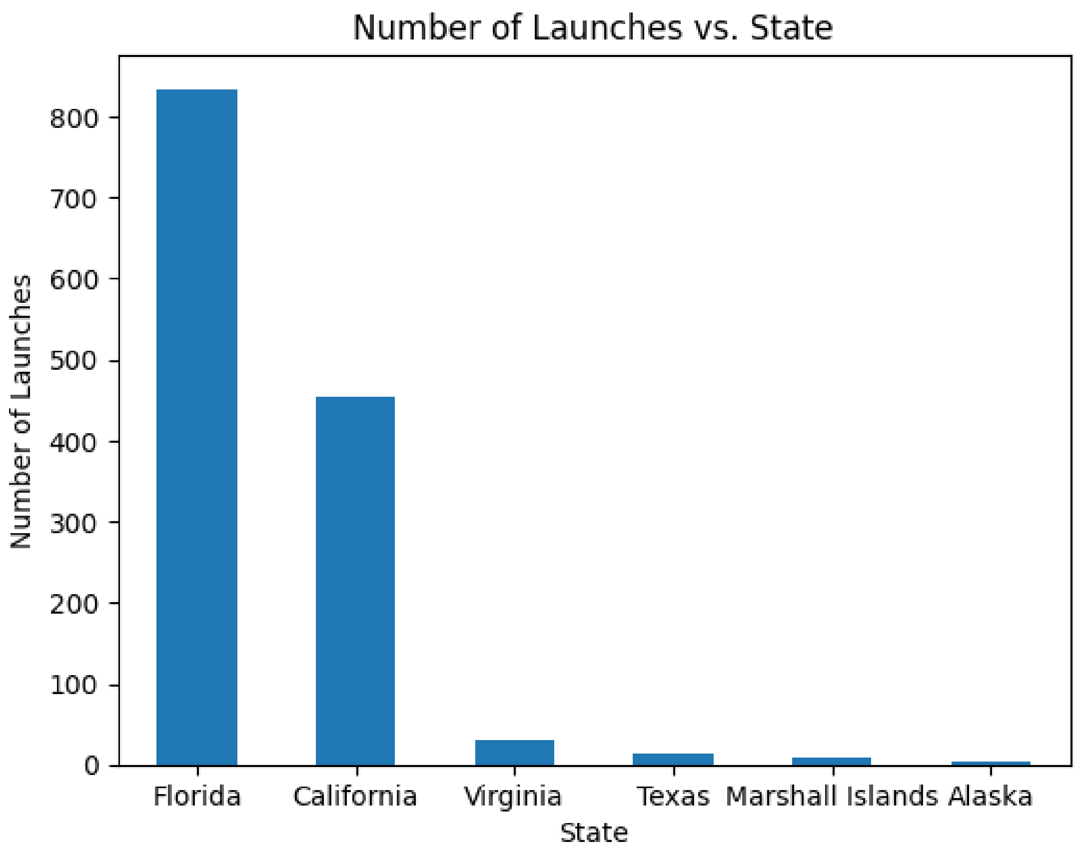
```
       'KARI', 'ESA', 'NASA', 'Boeing', 'ISAS', 'SRC', 'MITT', 'Lockheed',
       'AEB', 'Starsem', 'RVSN USSR', 'EER', 'General Dynamics',
       'Martin Marietta', 'Yuzhmash', 'Douglas', 'ASI', 'US Air Force',
       'CNES', 'CECLES', 'RAE', 'UT', 'OKB-586', 'AMBA', "Armée de l'Air",
       'US Navy'], dtype=object)
```

[273]:

```
df_US = space[space.US_or_Not == "USA"]
a = df_US.State.value_counts()
a.plot.bar(rot=0, xlabel="State", ylabel="Number of Launches", title="Number of␣
 ↪Launches vs. State")
```

[273]:

```
<Axes: title={'center': 'Number of Launches vs. State'}, xlabel='State',
ylabel='Number of Launches'>
```



[274]:

```
space.Country.unique()
```

[274]:

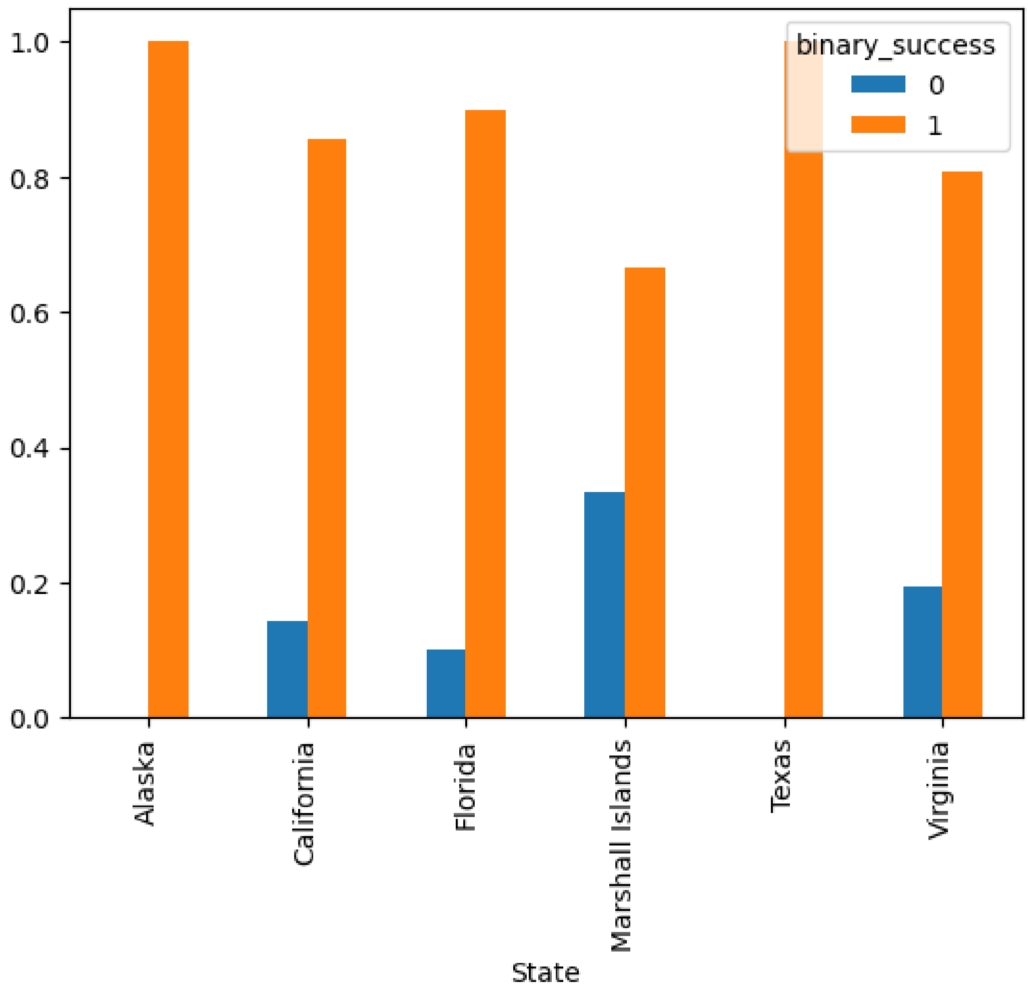
```
array([' USA', ' China', ' Kazakhstan', ' Japan', ' Israel',
       ' New Zealand', ' Russia', ' Shahrud Missile Test Site', ' France',
       ' Iran', ' India', 'USA', ' Yellow Sea', ' North Korea',
       ' Pacific Missile Range Facility', ' Pacific Ocean',
```

```
' South Korea', ' Barents Sea', ' Brazil', ' Gran Canaria',
' Kenya', ' Australia'], dtype=object)
```

Some analysis on the further data. I got some feedback from my friend and he said the plot (two below) is not an accurate representation of success. Yes, Florida has had the most successful launches but we also have to take into account that Florida also had the most launches. So, I decided to go with the crosstab format because it shows the actual percentage of success and failure per launch.

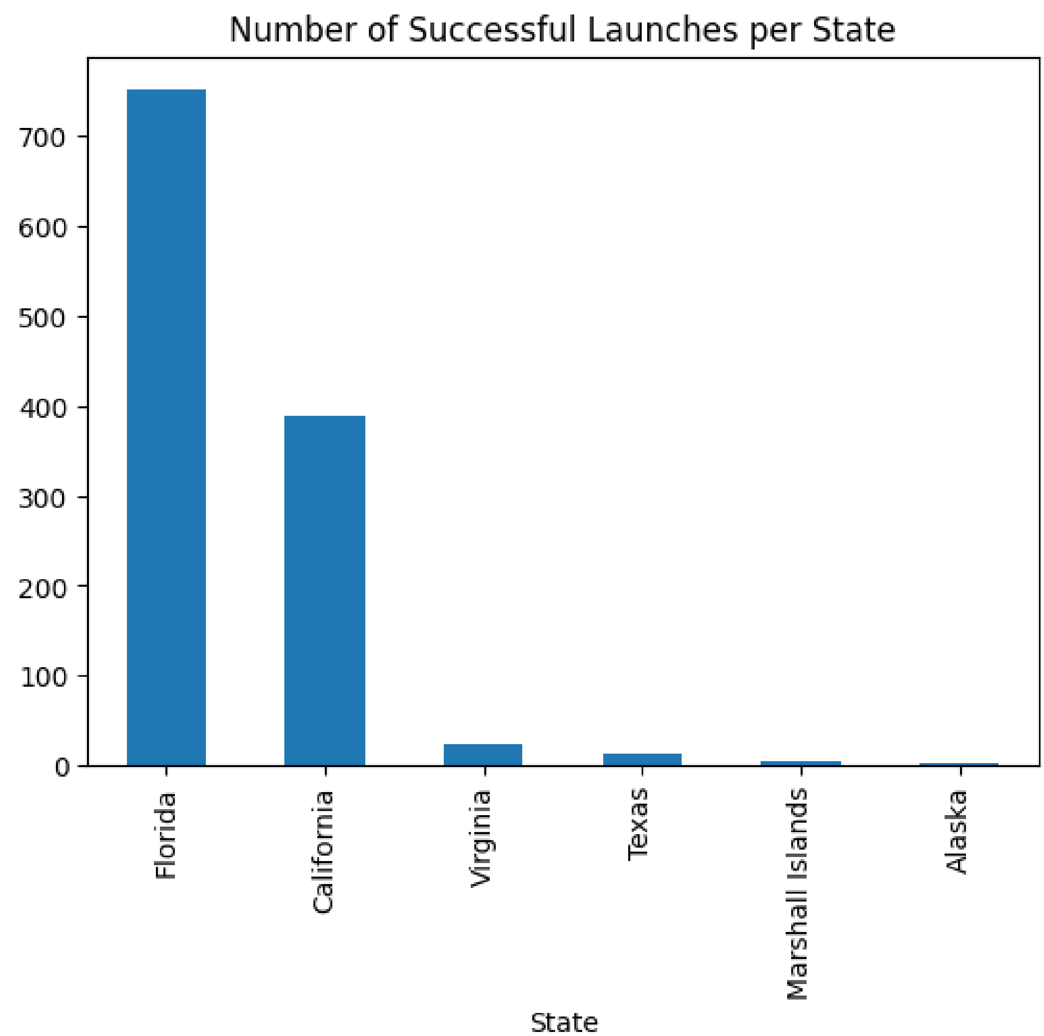
[275]:
```
a = pd.crosstab(df_US.State, df_US.binary_success, normalize=True)
success_given_state = a.divide(
    a.sum(axis=1),axis=0
)
success_given_state.plot.bar()
```

[275]: <Axes: xlabel='State'>



[276]:
```python
df_US.groupby("State")["binary_success"].sum().sort_values(ascending=False).
↪plot.bar(title = "Number of Successful Launches per State")
```

[276]:
```
<Axes: title={'center': 'Number of Successful Launches per State'},
xlabel='State'>
```



[277]:
```python
joint_success_state = pd.crosstab(
    df_US["State"], df_US["binary_success"]
)

joint_success_state
```

[277]:
```
binary_success     0    1
State
Alaska             0    3
```

```
California         65  389
Florida            84  750
Marshall Islands    3    6
Texas               0   13
Virginia            6   25
```

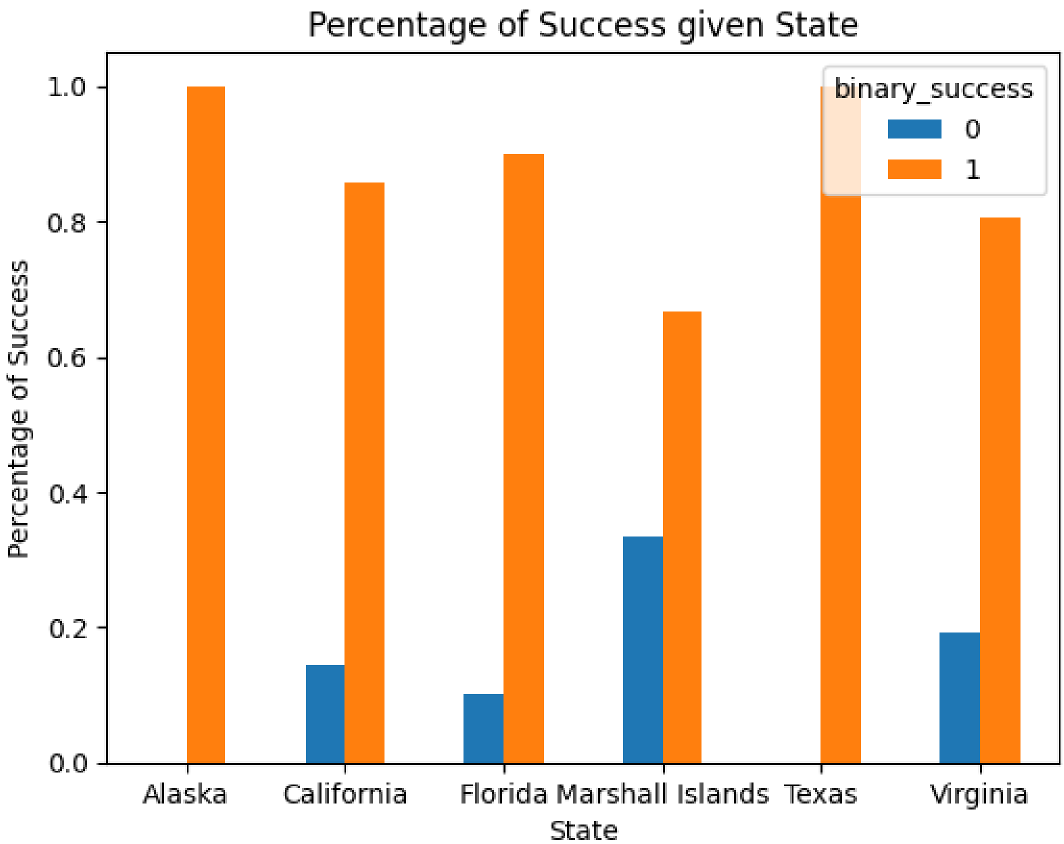
success given state

[278]:
```python
success_given_state = joint_success_state.divide(
    joint_success_state.sum(axis=1),axis=0
)
success_given_state
```

[278]:
```
binary_success            0         1
State
Alaska             0.000000  1.000000
California         0.143172  0.856828
Florida            0.100719  0.899281
Marshall Islands   0.333333  0.666667
Texas              0.000000  1.000000
Virginia           0.193548  0.806452
```

[279]:
```python
success_given_state.plot.bar(rot=0, ylabel="Percentage of Success",␣
 ↪title="Percentage of Success given State")
```

[279]:
```
<Axes: title={'center': 'Percentage of Success given State'}, xlabel='State',
ylabel='Percentage of Success'>
```




```
[280]: px.bar(success_given_state, labels = {"State": "State", "value": "Percentage"},
              title = "Conditional Distributions for Success of a Rocket Launch given␣
        ↪USA or not",
              category_orders={"binary_success": ["Success", "Failure"]})
```

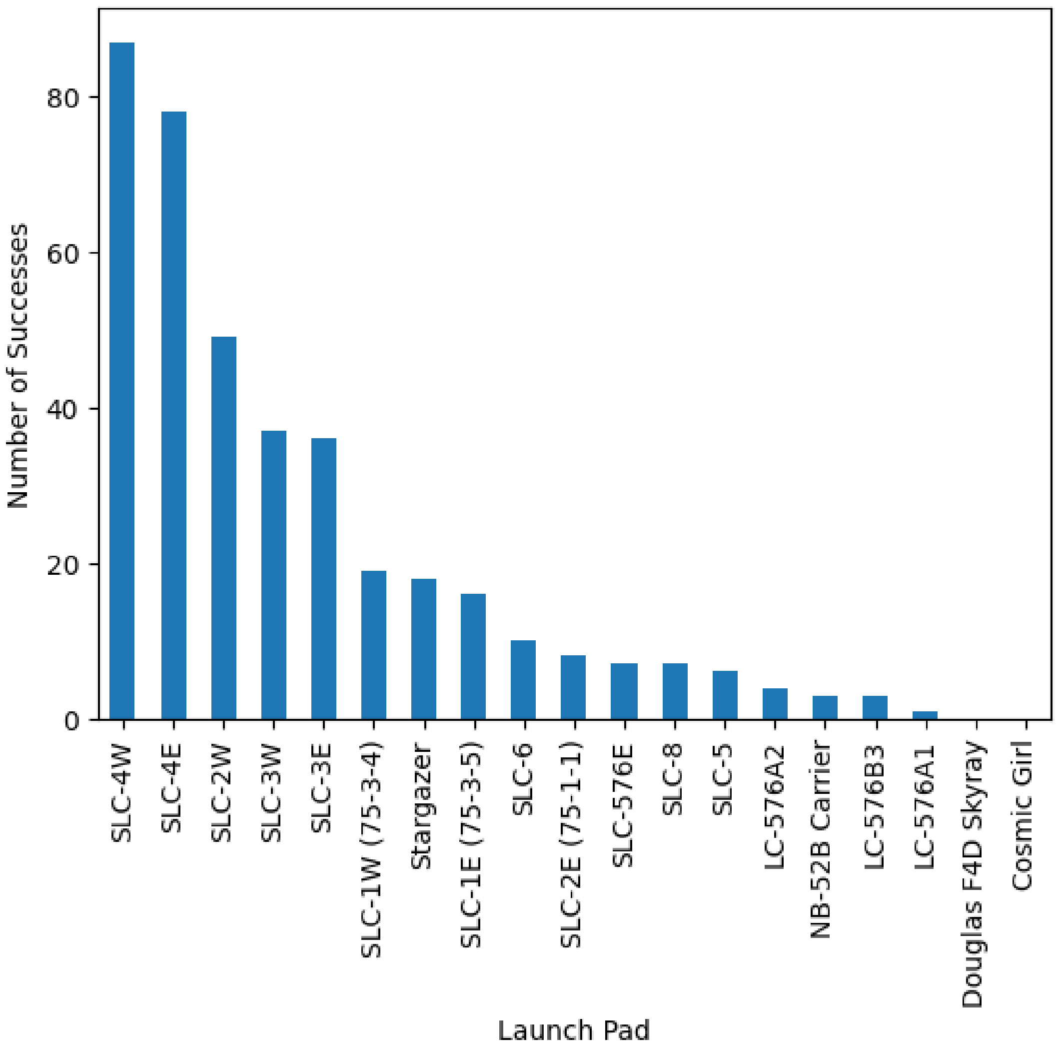
```
[281]: df_US.State.unique()
```

```
[281]: array(['Florida', 'Texas', 'Virginia', 'California', 'Marshall Islands',
              'Alaska'], dtype=object)
```

```
[282]: df_California = df_US[df_US.State == "California"]
```

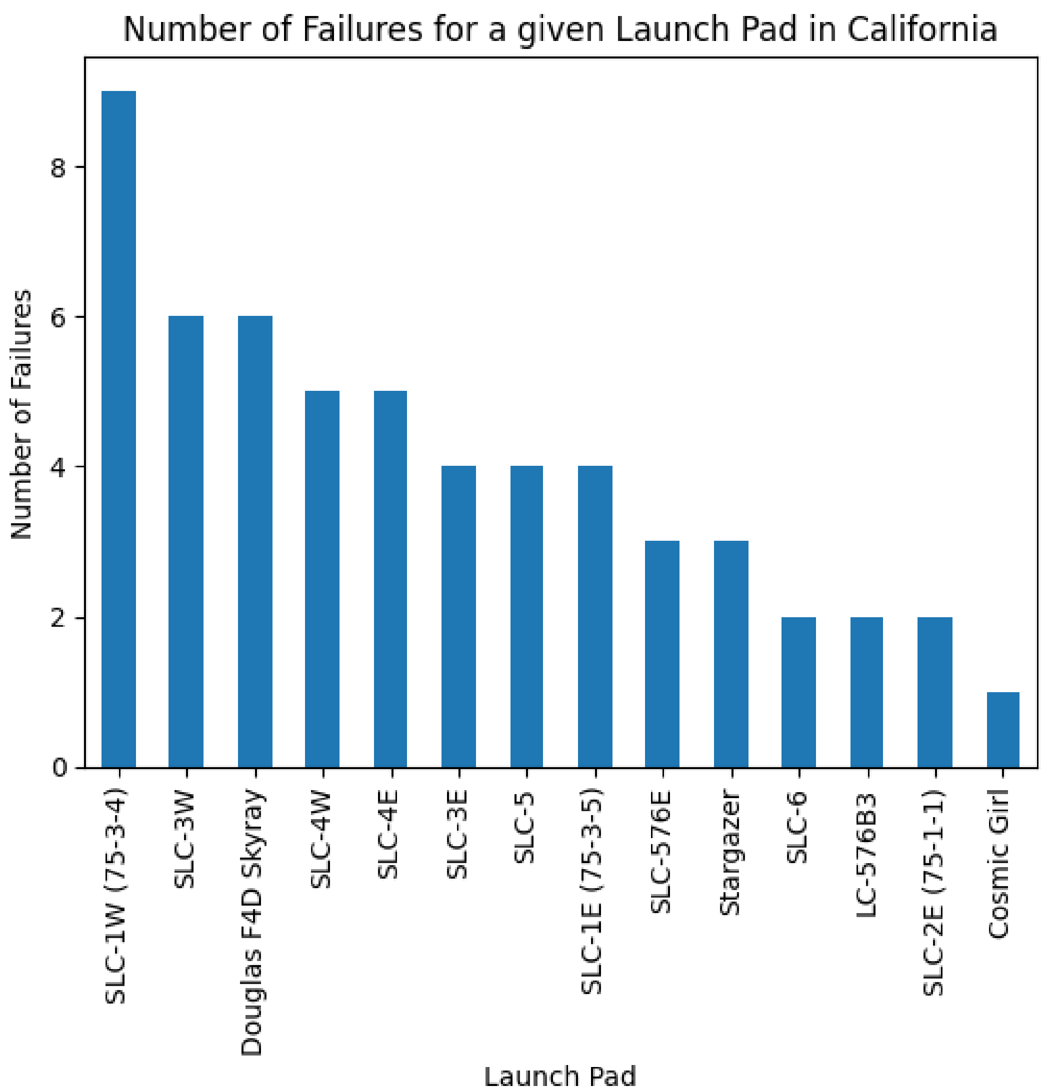
```
[283]: df_California.groupby("Launch Pad")["binary_success"].sum().
        ↪sort_values(ascending=False).plot.bar(xlabel="Launch Pad", ylabel="Number of␣
        ↪Successes")
```

```
[283]: <Axes: xlabel='Launch Pad', ylabel='Number of Successes'>
```



[284]:
```
df_California_failures = df_California[df_California["Status Mission"] ==␣
    ↪"Failure"]
df_California_failures["Launch Pad"].value_counts().plot.bar(xlabel = "Launch␣
    ↪Pad", ylabel="Number of Failures", title="Number of Failures for a given␣
    ↪Launch Pad in California")
```

[284]:
```
<Axes: title={'center': 'Number of Failures for a given Launch Pad in
California'}, xlabel='Launch Pad', ylabel='Number of Failures'>
```



3 TODO: For later just some ideas

- Price and Success just maybe ill do it later!!
- Most Expensive Rocket
- Company to Spend the Most
- Most Used Rocket, Most Used Launch Center
-

[285]:
```
space.head()
```

[285]:
```
   Unnamed: 0.1  Unnamed: 0 Company Name  \
0             0           0       SpaceX
1             1           1         CASC
```

```
2             2            2       SpaceX
3             3            3    Roscosmos
4             4            4          ULA

                                                   Location  \
0         LC-39A, Kennedy Space Center, Florida, USA
1  Site 9401 (SLS-2), Jiuquan Satellite Launch Ce…
2                      Pad A, Boca Chica, Texas, USA
3       Site 200/39, Baikonur Cosmodrome, Kazakhstan
4           SLC-41, Cape Canaveral AFS, Florida, USA

                        Datum                                     Detail  \
0  Fri Aug 07, 2020 05:12 UTC  Falcon 9 Block 5 | Starlink V1 L9 & BlackSky
1  Thu Aug 06, 2020 04:01 UTC           Long March 2D | Gaofen-9 04 & Q-SAT
2  Tue Aug 04, 2020 23:57 UTC            Starship Prototype | 150 Meter Hop
3  Thu Jul 30, 2020 21:25 UTC  Proton-M/Briz-M | Ekspress-80 & Ekspress-103
4  Thu Jul 30, 2020 11:50 UTC                    Atlas V 541 | Perseverance

 Status Rocket  Rocket Status Mission  Status  …         Launch Pad  \
0  StatusActive   50.0          Success  Active  …             LC-39A
1  StatusActive  29.75          Success  Active  …  Site 9401 (SLS-2)
2  StatusActive     NaN         Success  Active  …              Pad A
3  StatusActive   65.0          Success  Active  …        Site 200/39
4  StatusActive  145.0          Success  Active  …             SLC-41

                     Launch Center      Country    State  \
0                 Kennedy Space Center          USA  Florida
1   Jiuquan Satellite Launch Center        China     None
2                           Boca Chica          USA    Texas
3                  Baikonur Cosmodrome   Kazakhstan     None
4                    Cape Canaveral AFS          USA  Florida

             Vehicle Day of the Week Month Day Agency Type Satellite Count
0    Falcon 9 Block 5             Fri   Aug  07  Commercial               2
1       Long March 2D             Thu   Aug  06  Government               2
2  Starship Prototype             Tue   Aug  04  Commercial               1
3     Proton-M/Briz-M             Thu   Jul  30  Government               2
4         Atlas V 541             Thu   Jul  30  Commercial               1

[5 rows x 23 columns]
```

[286]:
```
space["Status Mission"].value_counts()
```

[286]:
```
Success             3879
Failure              339
Partial Failure      102
Prelaunch Failure      4
```

```
Name: Status Mission, dtype: int64
```

[287]:
```
space["Day"].astype(int)
```

[287]:
```
0        7
1        6
2        4
3       30
4       30
        ..
4319     5
4320     1
4321     6
4322     3
4323     4
Name: Day, Length: 4324, dtype: int64
```

[288]:
```
import plotly.express as px
ds = space["Status Mission"].value_counts().reset_index()

ds
```

[288]:
```
               index  Status Mission
0            Success            3879
1            Failure             339
2    Partial Failure             102
3  Prelaunch Failure               4
```

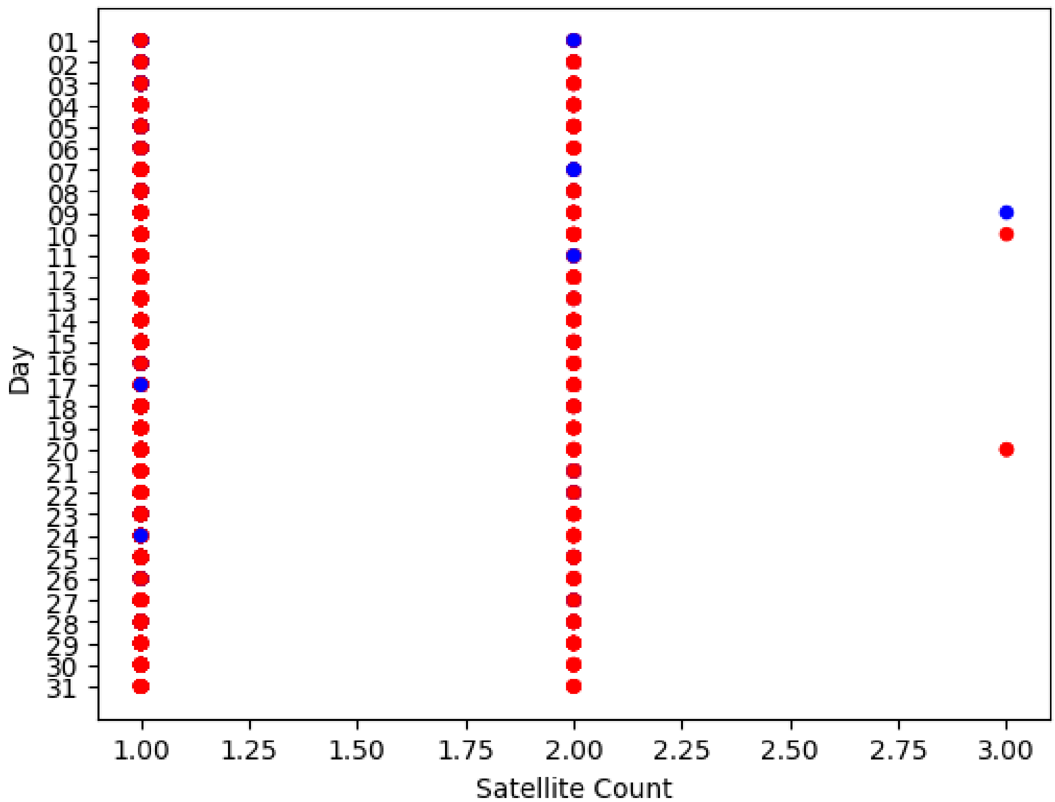
[289]:
```
px.pie(ds, values = "Status Mission", names="index" )
```

[290]:
```
space["binary_success_label"] = space["binary_success"].map({0: "Failure", 1:␣
↪"Success"})
p = space["binary_success_label"].value_counts().reset_index()
p
px.pie(p, values = "binary_success_label", names="index")
```

[291]:
```
space_byDay = space.sort_values(by="Day", ascending=False)
colors = space["binary_success"].map({
    0: "blue",
    1: "red"
})

space_byDay.plot.scatter(
    x = "Satellite Count", y = "Day", c=colors
)
```

[291]: <Axes: xlabel='Satellite Count', ylabel='Day'>



[292]:
```
def successOrnot(c):
  fail = ["Failure"]
  if c in fail:
    return "Failure"
  else:
    return "Success"

space["successornot"] = space["Status Mission"].map(successOrnot)
```

Focusing on three features: Agency Type, Satellite Count for sake of the graph

[293]:
```
colors = space["successornot"].map({
    "Success": "blue",
    "Failure": "red"
})

px.scatter(space, x="Country", y="Satellite Count", color = colors, opacity=.2)
```

[294]:
```
# df_California.groupby("Launch Pad")["binary_success"].sum().
↪sort_values(ascending=False).plot.bar(xlabel="Launch Pad", ylabel="Number of␣
↪Successes")
```

```
df_success = space.groupby(['Country', "Satellite Count"])["binary_success"].
    ↪sum().reset_index()
px.scatter(df_success, x = "Country", y="Satellite Count", color =␣
    ↪"binary_success")
```

[295]:
```
space["binary_success"].value_counts()
```

[295]:
```
1    3879
0     445
Name: binary_success, dtype: int64
```

Making a dataframe of failed launches. I did this so I can see what attributes can lead to a failure, used for machine learning later.

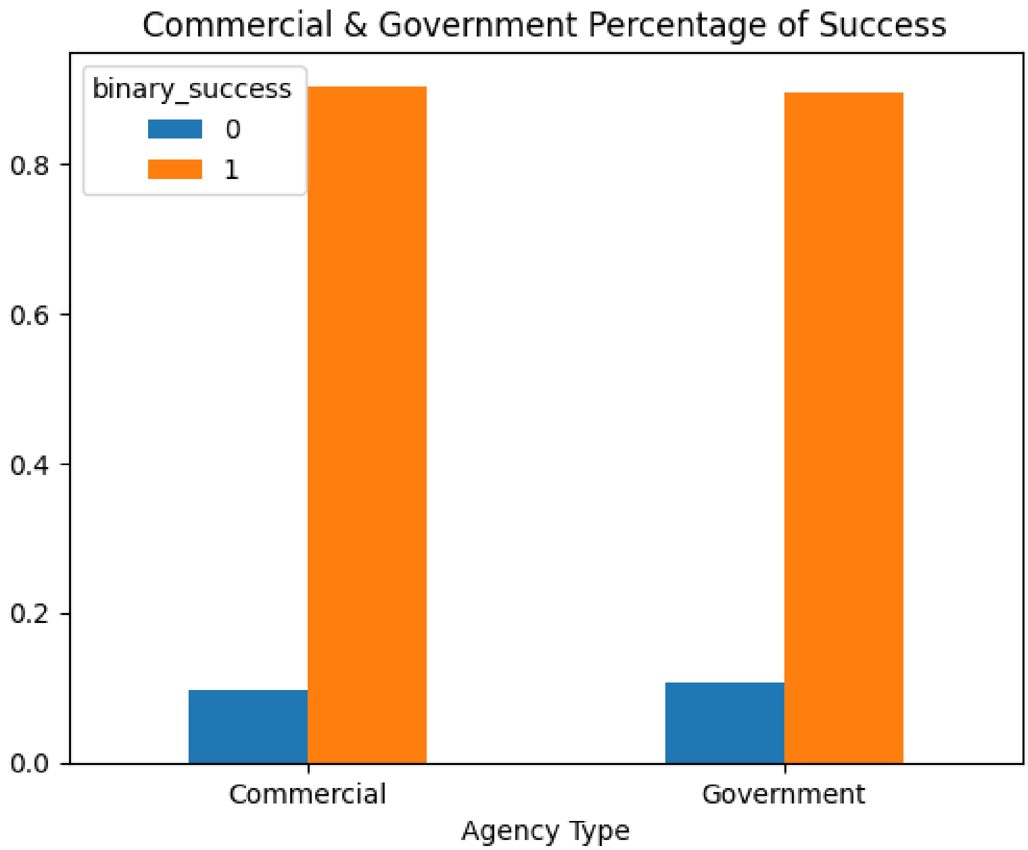
[296]:
```
failCount = space[space["binary_success_label"] == "Failure"]
failCount.reset_index().head()
failCount.shape
```

[296]:
```
(445, 25)
```

Percentage of success per agency type, better represented as a percentage than a count, as there are significantly more government launches.

[297]:
```
a = pd.crosstab(space["Agency Type"], space["binary_success"], normalize=True)
success_given_agency = a.divide(
    a.sum(axis=1),axis=0
)
success_given_agency.plot.bar(rot=0, title="Commercial & Government Percentage␣
    ↪of Success")
success_given_agency
```

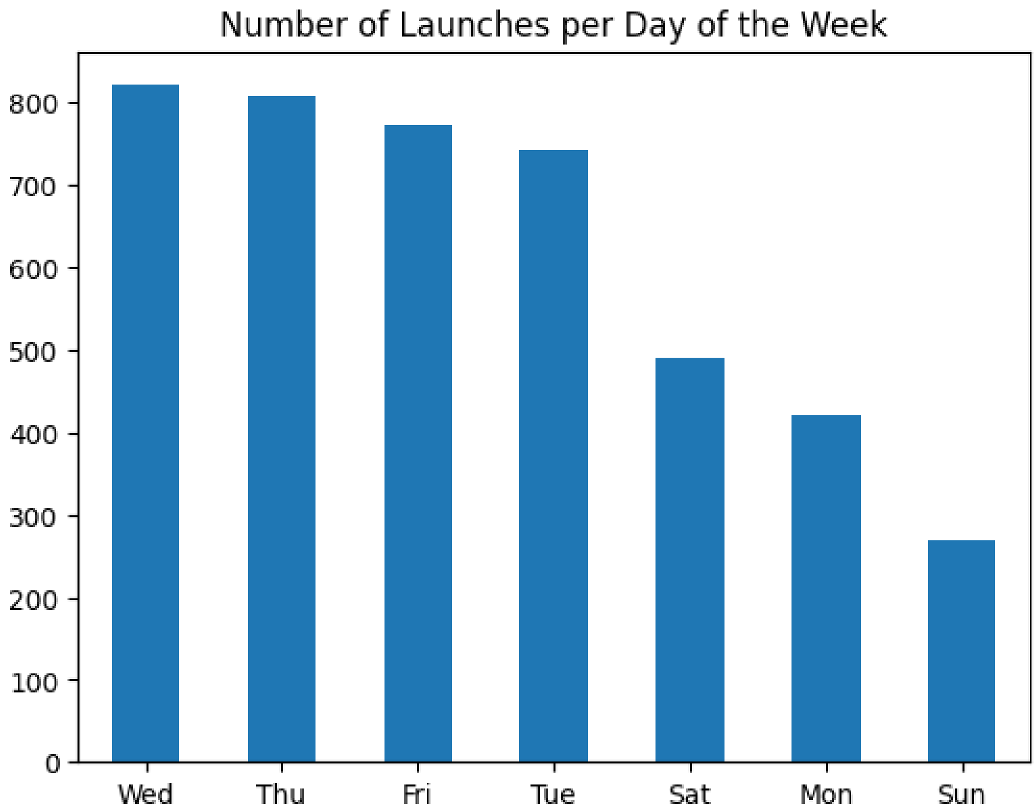
[297]:
```
binary_success         0         1
Agency Type
Commercial      0.096795  0.903205
Government      0.106261  0.893739
```



Total number of launches per day of the week for both success and failures, wanted to see if there was a trend.

[298]:
```
space["Day of the Week"].value_counts().plot.bar(rot=0, title=("Number of␣
↪Launches per Day of the Week"))
```

[298]: `<Axes: title={'center': 'Number of Launches per Day of the Week'}>`



Now this is unsuccessful launches per day of the week.

[299]:
```
dayData = failCount["Day of the Week"].value_counts()
px.bar(failCount, x=dayData.index, y = dayData.values, title= "Number of Failed␣
↪Launches per Day of the Week")
```

Number of failed launches per month

[300]:
```
monthData = failCount["Month"].value_counts()
px.bar(failCount, x=monthData.index, y = monthData.values, title="Number of␣
↪Failed Launches per month")
```

Number of failed Launches per Country

[301]:
```
countryData = failCount["Country"].value_counts()
px.bar(failCount, x=countryData.index, y = countryData.values, title="Number of␣
↪Failed Launches per Country")
```

4 Actual Start of my machine learning

[302]:
```
space['Country'] = space['Country'].str.strip()
```

Predicting mission status by using the features, month, country, and agency type.

[303]:
```
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()
```

[303]:
```
1    3815
0     509
dtype: int64
```

Getting the cv_scores for my model

[304]:
```
from sklearn.model_selection import cross_val_score
cv_scores = cross_val_score(pipeline20, xTT, yTT,
                            cv=10, scoring="accuracy")
cv_scores
cv_scores.mean()
```

[304]:
```
0.8115147335557268
```

Precision, recall, f1 score fir success and failure

[305]:
```python
success = (yTT == 1)

cv_scores = cross_val_score(pipeline20, xTT, success,
                cv=10, scoring="precision")

precision_success = cv_scores.mean()
precision_success

recall_success = cross_val_score(pipeline20, xTT, success,
                cv=10, scoring="recall").mean()
recall_success
f1score_success = cross_val_score(pipeline20, xTT, success,
                cv=10, scoring="f1").mean()

precision_success, recall_success, f1score_success
```

[305]: (0.8964957418599067, 0.893014598151256, 0.8946016140067016)

[306]:
```python
from sklearn.metrics import make_scorer, precision_score, recall_score
failure = (yTT == 0)
precision_scorer = make_scorer(precision_score, zero_division=1)
recall_scorer = make_scorer(recall_score, zero_division=1)

cv_scores = cross_val_score(pipeline20, xTT, failure,
                cv=10, scoring=precision_scorer)

precision_failure = cv_scores.mean()
precision_failure

recall_failure = cross_val_score(pipeline20, xTT, failure,
                cv=10, scoring=recall_scorer).mean()
recall_failure
f1score_failure = cross_val_score(pipeline20, xTT, failure,
                cv=10, scoring="f1").mean()

precision_failure, recall_failure, f1score_failure
```

[306]: (0.8, 0.0, 0.0)

[307]:
```python
from sklearn.model_selection import GridSearchCV

grid_search = GridSearchCV(
    pipeline20,
    param_grid={"kneighborsclassifier__n_neighbors": range(1, 50)},
    scoring="f1_macro",
    cv=10
)
```

```
grid_search.fit(xTT, yTT)
grid_search.best_params_
```

[307]:

```
{'kneighborsclassifier__n_neighbors': 2}
```

Best valu for k is the one that was passed in

[308]:

```
pd.DataFrame(grid_search.cv_results_).sort_values("rank_test_score").head(10)
```

[308]:

```
    mean_fit_time  std_fit_time  mean_score_time  std_score_time  \
1        0.013678      0.003728         0.074478        0.004534
0        0.012426      0.002227         0.073096        0.002955
9        0.014859      0.003564         0.081341        0.016656
2        0.011691      0.001248         0.071931        0.007054
3        0.012695      0.001599         0.074361        0.006428
28       0.011486      0.001334         0.074654        0.003456
29       0.010811      0.000186         0.072369        0.001586
30       0.012683      0.003739         0.078065        0.004515
31       0.011995      0.002559         0.075771        0.004371
32       0.011203      0.001177         0.075015        0.003850

   param_kneighborsclassifier__n_neighbors  \
1                                        2
0                                        1
9                                       10
2                                        3
3                                        4
28                                      29
29                                      30
30                                      31
31                                      32
32                                      33

                                      params  split0_test_score  \
1    {'kneighborsclassifier__n_neighbors': 2}           0.513811
0    {'kneighborsclassifier__n_neighbors': 1}           0.510381
9   {'kneighborsclassifier__n_neighbors': 10}           0.472594
2    {'kneighborsclassifier__n_neighbors': 3}           0.472594
3    {'kneighborsclassifier__n_neighbors': 4}           0.468059
28  {'kneighborsclassifier__n_neighbors': 29}           0.472594
29  {'kneighborsclassifier__n_neighbors': 30}           0.472594
30  {'kneighborsclassifier__n_neighbors': 31}           0.472594
31  {'kneighborsclassifier__n_neighbors': 32}           0.472594
32  {'kneighborsclassifier__n_neighbors': 33}           0.472594

    split1_test_score  split2_test_score  split3_test_score  \
```

1	0.507955	0.512444	0.513864
0	0.487114	0.497611	0.522476
9	0.494910	0.472594	0.472594
2	0.490588	0.472594	0.471951
3	0.486499	0.467405	0.471306
28	0.472594	0.472594	0.472594
29	0.472594	0.472594	0.472594
30	0.472594	0.472594	0.472594
31	0.472594	0.472594	0.472594
32	0.472594	0.472594	0.472594

	split4_test_score	split5_test_score	split6_test_score \
1	0.455107	0.474772	0.472527
0	0.464474	0.467633	0.467944
9	0.473171	0.473171	0.473171
2	0.473171	0.472527	0.473171
3	0.473171	0.472527	0.473171
28	0.473171	0.473171	0.473171
29	0.473171	0.473171	0.473171
30	0.473171	0.473171	0.473171
31	0.473171	0.473171	0.473171
32	0.473171	0.473171	0.473171

	split7_test_score	split8_test_score	split9_test_score	mean_test_score \
1	0.536626	0.517561	0.457643	0.496231
0	0.507861	0.519531	0.465338	0.491036
9	0.473171	0.473171	0.472527	0.475107
2	0.472527	0.473171	0.471883	0.474418
3	0.471883	0.473171	0.471883	0.472907
28	0.473171	0.473171	0.472527	0.472876
29	0.473171	0.473171	0.472527	0.472876
30	0.473171	0.473171	0.472527	0.472876
31	0.473171	0.473171	0.472527	0.472876
32	0.473171	0.473171	0.472527	0.472876

	std_test_score	rank_test_score
1	0.027036	1
0	0.022278	2
9	0.006607	3
2	0.005408	4
3	0.004935	5
28	0.000295	6
29	0.000295	6
30	0.000295	6
31	0.000295	6
32	0.000295	6

5 Model Discussion

Trying with new model, i could not get the cv.errors() loop to work.

[309]:
```
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)
```

[309]:
```
Pipeline(steps=[('columntransformer',
                 ColumnTransformer(remainder='passthrough',
                                   transformers=[('onehotencoder',
                                                  OneHotEncoder(),
                                                  ['Month', 'Day of the Week',
                                                   'Agency Type'])])),
                ('kneighborsclassifier', KNeighborsClassifier(n_neighbors=2))])
```

[310]:
```
a = pipeline0.predict(xT1)
pd.Series(a).value_counts()
```

[310]:
```
1    3648
0     676
dtype: int64
```

[311]:
```
from sklearn.model_selection import cross_val_score
cv_scores = cross_val_score(pipeline0, xT1, yT1,
                            cv=10, scoring="accuracy")
cv_scores
```

[311]:
```
array([0.84295612, 0.76674365, 0.78290993, 0.78060046, 0.74768519,
       0.77314815, 0.75231481, 0.7337963 , 0.77314815, 0.75925926])
```

[312]:
```
cv_scores.mean()
```

[312]:
```
0.771256201351467
```

[ ]:
```

```

Precision, recall, and f1 scores for my model: looking at successes

[313]:
```
success = (yT1 == 1)

cv_scores = cross_val_score(pipeline0, xT1, success,
                cv=10, scoring="precision")

precision_success = cv_scores.mean()
precision_success
```

[313]: 0.893022100699319

[314]:
```
recall_success = cross_val_score(pipeline0, xT1, success,
                cv=10, scoring="recall").mean()
recall_success
```

[314]: 0.8463471323157249

[315]:
```
f1score_success = cross_val_score(pipeline0, xT1, success,
                cv=10, scoring="f1").mean()

f1score_success
```

[315]: 0.868811052927353

Precision, recall, and f1 scores for my model: looking at failures

[316]:
```
from sklearn.metrics import make_scorer, precision_score
failure = (yT1 == 0)
precision_scorer = make_scorer(precision_score, zero_division=1)
cv_scores = cross_val_score(pipeline0, xT1, failure,
                cv=10, scoring=precision_scorer)

precision_failure = cv_scores.mean()
precision_failure
```

[316]: 0.45333333333333333

[317]:
```
recall_scorer = make_scorer(recall_score, zero_division=1)
recall_failure = cross_val_score(pipeline0, xT1, failure,
                cv=10, scoring=recall_scorer).mean()
recall_failure
```

[317]: 0.006717171717171717

[318]:
```
f1score_failure = cross_val_score(pipeline0, xT1, failure,
                cv=10, scoring="f1").mean()
```

```
f1score_failure
```

[318]: 0.012611111111111111

Finding best k, using gridsearch

[319]:
```
from sklearn.model_selection import GridSearchCV

grid_search = GridSearchCV(
    pipeline0,
    param_grid={"kneighborsclassifier__n_neighbors": range(1, 50)},
    scoring="f1_macro",
    cv=10
)

grid_search.fit(xT1, yT1)
grid_search.best_params_
```

[319]: {'kneighborsclassifier__n_neighbors': 4}

[320]:
```
pd.DataFrame(grid_search.cv_results_).sort_values("rank_test_score").head(10)
```

[320]:
```
    mean_fit_time  std_fit_time  mean_score_time  std_score_time  \
3        0.012066      0.001946         0.071622        0.001905
0        0.011490      0.001673         0.073404        0.005207
1        0.011660      0.001974         0.071506        0.002051
2        0.011698      0.000954         0.072269        0.003263
7        0.010824      0.000185         0.071117        0.003218
5        0.017564      0.001032         0.219423        0.072002
6        0.012239      0.003002         0.073807        0.002079
4        0.015841      0.002919         0.143229        0.067609
9        0.011536      0.000866         0.073979        0.005381
35       0.012916      0.001150         0.075969        0.002110

   param_kneighborsclassifier__n_neighbors  \
3                                        4
0                                        1
1                                        2
2                                        3
7                                        8
5                                        6
6                                        7
4                                        5
9                                       10
35                                      36

                                       params  split0_test_score  \
```

```
3    {'kneighborsclassifier__n_neighbors': 4}           0.484524
0    {'kneighborsclassifier__n_neighbors': 1}           0.484524
1    {'kneighborsclassifier__n_neighbors': 2}           0.509594
2    {'kneighborsclassifier__n_neighbors': 3}           0.487503
7    {'kneighborsclassifier__n_neighbors': 8}           0.490588
5    {'kneighborsclassifier__n_neighbors': 6}           0.490588
6    {'kneighborsclassifier__n_neighbors': 7}           0.490588
4    {'kneighborsclassifier__n_neighbors': 5}           0.490588
9   {'kneighborsclassifier__n_neighbors': 10}           0.490588
35  {'kneighborsclassifier__n_neighbors': 36}           0.472594

    split1_test_score  split2_test_score  split3_test_score  \
3            0.504921           0.466091           0.464771
0            0.506690           0.491596           0.478687
1            0.500953           0.494762           0.509158
2            0.492716           0.470660           0.467405
7            0.472594           0.472594           0.493804
5            0.472594           0.471306           0.468712
6            0.472594           0.472594           0.472594
4            0.472594           0.472594           0.472594
9            0.472594           0.472594           0.472594
35           0.472594           0.472594           0.472594

    split4_test_score  split5_test_score  split6_test_score  \
3            0.469287           0.513209           0.513209
0            0.475051           0.453856           0.483782
1            0.453323           0.464372           0.446969
2            0.473171           0.471883           0.471883
7            0.473171           0.473171           0.473171
5            0.472527           0.471883           0.471883
6            0.473171           0.473171           0.473171
4            0.472527           0.473171           0.473171
9            0.471883           0.473171           0.473171
35           0.473171           0.473171           0.473171

    split7_test_score  split8_test_score  split9_test_score  mean_test_score  \
3            0.471883           0.494844           0.469939         0.485268
0            0.449682           0.519342           0.487010         0.483022
1            0.447404           0.497340           0.496774         0.482065
2            0.473171           0.495971           0.471236         0.477560
7            0.473171           0.473171           0.472527         0.476796
5            0.473171           0.495971           0.471236         0.475987
6            0.473171           0.473171           0.472527         0.474675
4            0.472527           0.473171           0.472527         0.474547
9            0.473171           0.473171           0.472527         0.474546
35           0.473171           0.473171           0.472527         0.472876
```

```
    std_test_score  rank_test_score
3         0.018701                1
0         0.020040                2
1         0.024560                3
2         0.009804                4
7         0.007738                5
5         0.008804                6
6         0.005312                7
4         0.005354                8
9         0.005362                9
35        0.000295               10
```

New precision and new recall for both success and failure

[321]:
```python
new_precision = cross_val_score(
    grid_search.best_estimator_,
    xT1, success,
    scoring="precision",
    cv=10).mean()

new_recall = cross_val_score(
    grid_search.best_estimator_,
    xT1, success,
    scoring="recall",
    cv=10).mean()

new_precision, new_recall
```

[321]:
```
(0.8973334946288993, 0.9847911505367752)
```

[322]:
```python
precision_success, recall_success
```

[322]:
```
(0.893022100699319, 0.8463471323157249)
```

[323]:
```python
new_precision_failure = cross_val_score(
    grid_search.best_estimator_,
    xT1, failure,
    scoring=precision_scorer,
    cv=10).mean()

new_recall_failure = cross_val_score(
    grid_search.best_estimator_,
    xT1, failure,
    scoring=recall_scorer,
    cv=10).mean()

new_precision_failure, new_recall_failure
```

[323]: (0.9199999999999999, 0.0022222222222222222)

This is the original precision and failure for model 2

[324]:
```
precision_failure, recall_failure
```

[324]: (0.4533333333333333, 0.006717171717171717)

[325]:
```
space.columns
```

[325]:
```
Index(['Unnamed: 0.1', 'Unnamed: 0', 'Company Name', 'Location', 'Datum',
       'Detail', 'Status Rocket', ' Rocket', 'Status Mission', 'Status',
       'US_or_Not', 'binary_success', 'failureCount', 'Launch Pad',
       'Launch Center', 'Country', 'State', 'Vehicle', 'Day of the Week',
       'Month', 'Day', 'Agency Type', 'Satellite Count',
       'binary_success_label', 'successornot'],
      dtype='object')
```

**Start of new model, checking how day of the week and month are related.

[326]:
```
colors = space["binary_success"].map({
    0: "blue",
    1: "red"
})

px.scatter(space, x="Month", y="Day of the Week", color = colors, opacity = 0.
 ↪2, title="Success/Failure of Day in the Week vs. Month")
```

[327]:
```
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)
```

[327]:
```
Pipeline(steps=[('columntransformer',
                 ColumnTransformer(remainder='passthrough',
                                   transformers=[('onehotencoder',
```

```
                                            OneHotEncoder(),
                                            ['Month',
                                             'Day of the Week'])])),
                ('kneighborsclassifier', KNeighborsClassifier(n_neighbors=2))])
```

[328]:
```
a = pipeline_new.predict(xT_new)
pd.Series(a).value_counts()
```

[328]:
```
Success    3749
Failure     575
dtype: int64
```

[329]:
```
from sklearn.model_selection import cross_val_score
cv_scores = cross_val_score(pipeline_new, xT_new, yT_new,
                            cv=10, scoring="accuracy")
cv_scores.mean()
```

[329]:
```
0.7960006629030877
```

[330]:
```
from sklearn.model_selection import GridSearchCV

grid_search = GridSearchCV(
    pipeline_new,
    param_grid={"kneighborsclassifier__n_neighbors": range(1, 50)},
    scoring="f1_macro",
    cv=10
)

grid_search.fit(xT_new, yT_new)
grid_search.best_params_
```

[330]:
```
{'kneighborsclassifier__n_neighbors': 2}
```

The number of k nearest neighbors is the same as my original model.

6 Making chloropeth plots for fun

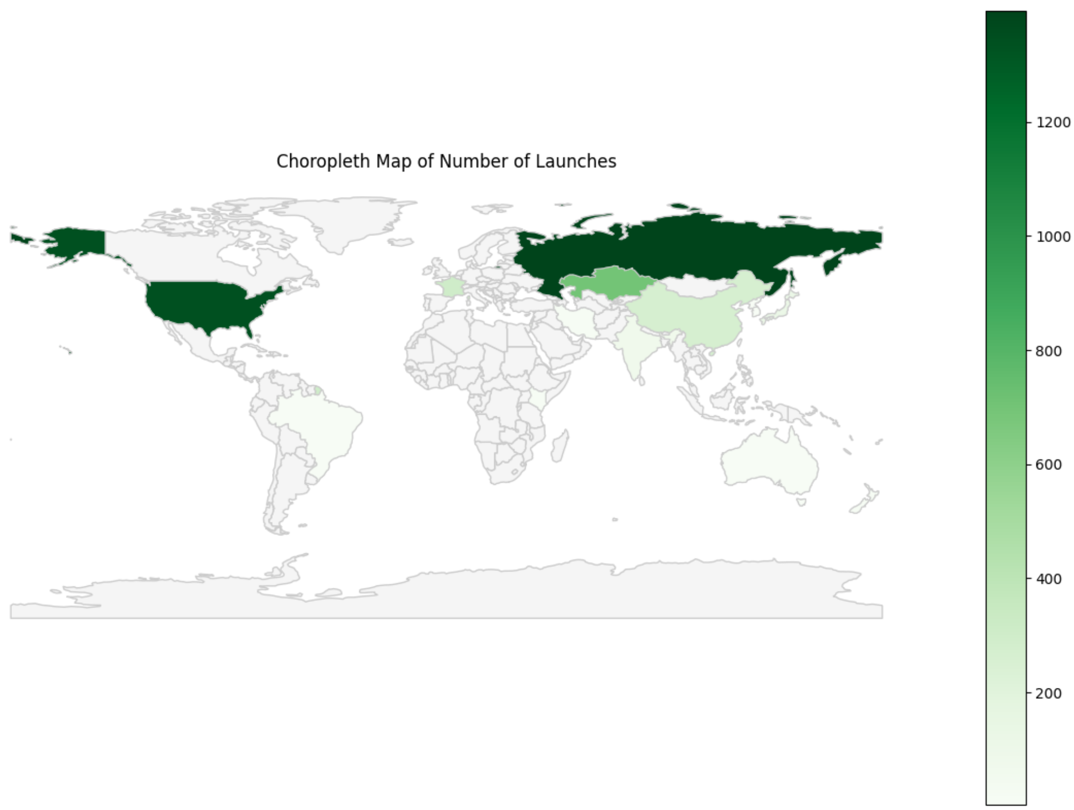
[331]:
```
world = gpd.read_file(gpd.datasets.get_path('naturalearth_lowres'))  # this is␣
↪a part of geopandas set
geo_df = world.merge(df1, how='left', left_on='name', right_on='Country') # use␣
↪the df that I created
fig, ax = plt.subplots(1, 1, figsize=(15, 10))
world.plot(ax=ax, color='whitesmoke', edgecolor='0.8')  # plotting the world
geo_df.plot(column='freq', cmap='Greens', linewidth=0.8, ax=ax, edgecolor='0.
↪8', legend=True) # use the frequency from the column that I created
```

```
# Add labels and title
ax.set_title('Choropleth Map of Number of Launches')
ax.set_axis_off()

plt.show()
```

```
<ipython-input-331-aa4bb7997231>:1: FutureWarning:

The geopandas.dataset module is deprecated and will be removed in GeoPandas 1.0.
You can get the original 'naturalearth_lowres' data from
https://www.naturalearthdata.com/downloads/110m-cultural-vectors/.
```



[332]:
```
world_filepath = gpd.datasets.get_path('naturalearth_lowres')
world = gpd.read_file(world_filepath)
world.head()
```

```
<ipython-input-332-794e9e622c9b>:1: FutureWarning:

The geopandas.dataset module is deprecated and will be removed in GeoPandas 1.0.
You can get the original 'naturalearth_lowres' data from
https://www.naturalearthdata.com/downloads/110m-cultural-vectors/.
```

[332]:

```
         pop_est      continent                      name iso_a3  gdp_md_est  \
0     889953.0        Oceania                      Fiji    FJI        5496
1   58005463.0         Africa                  Tanzania    TZA       63177
2     603253.0         Africa                 W. Sahara    ESH         907
3   37589262.0  North America                    Canada    CAN     1736425
4  328239523.0  North America  United States of America    USA    21433226

                                            geometry
0  MULTIPOLYGON (((180.00000 -16.06713, 180.00000…
1  POLYGON ((33.90371 -0.95000, 34.07262 -1.05982…
2  POLYGON ((-8.66559 27.65643, -8.66512 27.58948…
3  MULTIPOLYGON (((-122.84000 49.00000, -122.9742…
4  MULTIPOLYGON (((-122.84000 49.00000, -120.0000…
```

[333]:

```
world = gpd.read_file(gpd.datasets.get_path('naturalearth_lowres'))

# Merge the GeoDataFrame with your DataFrame
geo_df = world.merge(df1, how='left', left_on='name', right_on='Country')␣
 ↪#match the country w iso3 in new df

# Create the choropleth map using Plotly Express
fig = px.choropleth(
    geo_df,
    locations='iso_a3',  # for natrualearth_lowres there are iso_a3 which are␣
 ↪the code names and name which is the actual name
    color='freq',
    color_continuous_scale='sunset',
    projection='natural earth',
    title='Choropleth Map of Number of Launches'
)


# Show the map
fig.show()
```

```
<ipython-input-333-222bbeb83844>:1: FutureWarning:

The geopandas.dataset module is deprecated and will be removed in GeoPandas 1.0.
You can get the original 'naturalearth_lowres' data from
https://www.naturalearthdata.com/downloads/110m-cultural-vectors/.
```

[334]:

```
space.head()
```

[334]:
```
   Unnamed: 0.1  Unnamed: 0 Company Name  \
0             0           0       SpaceX
1             1           1         CASC
2             2           2       SpaceX
3             3           3    Roscosmos
4             4           4          ULA

                                            Location  \
0         LC-39A, Kennedy Space Center, Florida, USA
1  Site 9401 (SLS-2), Jiuquan Satellite Launch Ce…
2                       Pad A, Boca Chica, Texas, USA
3       Site 200/39, Baikonur Cosmodrome, Kazakhstan
4           SLC-41, Cape Canaveral AFS, Florida, USA

                      Datum                                        Detail  \
0  Fri Aug 07, 2020 05:12 UTC  Falcon 9 Block 5 | Starlink V1 L9 & BlackSky
1  Thu Aug 06, 2020 04:01 UTC           Long March 2D | Gaofen-9 04 & Q-SAT
2  Tue Aug 04, 2020 23:57 UTC            Starship Prototype | 150 Meter Hop
3  Thu Jul 30, 2020 21:25 UTC  Proton-M/Briz-M | Ekspress-80 & Ekspress-103
4  Thu Jul 30, 2020 11:50 UTC                    Atlas V 541 | Perseverance

  Status Rocket  Rocket Status Mission  Status  …     Country    State  \
0  StatusActive   50.0         Success  Active  …         USA  Florida
1  StatusActive  29.75         Success  Active  …       China     None
2  StatusActive     NaN        Success  Active  …         USA    Texas
3  StatusActive   65.0         Success  Active  …  Kazakhstan     None
4  StatusActive  145.0         Success  Active  …         USA  Florida

              Vehicle Day of the Week Month Day Agency Type Satellite Count  \
0    Falcon 9 Block 5             Fri   Aug  07  Commercial               2
1       Long March 2D             Thu   Aug  06  Government               2
2  Starship Prototype             Tue   Aug  04  Commercial               1
3     Proton-M/Briz-M             Thu   Jul  30  Government               2
4         Atlas V 541             Thu   Jul  30  Commercial               1

  binary_success_label successornot
0              Success      Success
1              Success      Success
2              Success      Success
3              Success      Success
4              Success      Success

[5 rows x 25 columns]
```

[335]:
```python
df_rockets.head()
```

[335]:

	Rocket Name	Missions	Successes	Partial Failures	Failures
0	Alpha	3	1	1	1
1	Amur	0	0	0	0
2	Angara 1	3	3	0	0
3	Angara A5	3	2	0	1
4	Antares	18	17	0	1

	Success Streak	Success Rate	Accurate Success Rate	Failure Rate
0	1	50.0%	33.333333	0.666667
1	0	0	0.000000	0.000000
2	3	100%	100.000000	0.000000
3	0	66.7%	66.666667	0.333333
4	13	94.4%	94.444444	0.055556

	Total Failures
0	2
1	0
2	0
3	1
4	1