IBM data science capstone project with python code

Contents:

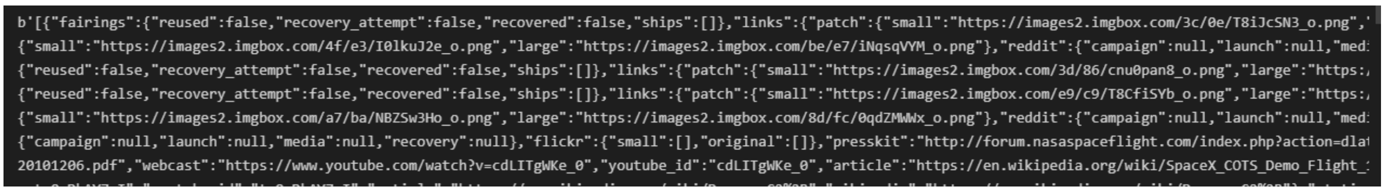
1/ Collecting data by SpaceX API

Now let's start requesting rocket launch data from SpaceX API with the following URL:

```
spacex_url="https://api.spacexdata.com/v4/launches/past"
response = requests.get(spacex_url)
```

#Check the content of the response

```
print(response.content)
```



```
b'[{"fairings":{"reused":false,"recovery_attempt":false,"recovered":false,"ships":[]},"links":{"patch":{"small":"https://images2.imgbox.com/3c/0e/T8iJcSN3_o.png",'
{"small":"https://images2.imgbox.com/4f/e3/I0lkuJ2e_o.png","large":"https://images2.imgbox.com/be/e7/iNqsqVYM_o.png"},"reddit":{"campaign":null,"launch":null,"med
{"reused":false,"recovery_attempt":false,"recovered":false,"ships":[]},"links":{"patch":{"small":"https://images2.imgbox.com/3d/86/cnu0pan8_o.png","large":"https:/
{"reused":false,"recovery_attempt":false,"recovered":false,"ships":[]},"links":{"patch":{"small":"https://images2.imgbox.com/e9/c9/T8CfiSYb_o.png","large":"https:/
{"small":"https://images2.imgbox.com/a7/ba/NBZSw3Ho_o.png","large":"https://images2.imgbox.com/8d/fc/0qdZMWWx_o.png"},"reddit":{"campaign":null,"launch":null,"med
{"campaign":null,"launch":null,"media":null,"recovery":null},"flickr":{"small":[],"original":[]},"presskit":"http://forum.nasaspaceflight.com/index.php?action=dla
20101206.pdf","webcast":"https://www.youtube.com/watch?v=cdLITgWKe_0","youtube_id":"cdLITgWKe_0","article":"https://en.wikipedia.org/wiki/SpaceX_COTS_Demo_Flight_
```

Task 1: Request and parse the SpaceX launch data using the GET request

To make the requested JSON results more consistent, we will use the following static response object for this project:

```
static_json_url='https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/API_call_spacex_api.json'
```

We should see that the request was successful with the 200 status response code

```
response.status_code →kết quả: 200
```

Now we decode the response content as a Json using .json() and turn it into a Pandas dataframe using .json_normalize()

convert the json result into a dataframe → pd.json_normalize

```
data = pd.json_normalize(response.json())
```

Using the dataframe data print the first 5 rows

# Get the head of the dataframe

```
data.head(5)
```

You will notice that a lot of the data are IDs. For example, the rocket column has no information about the rocket just an identification number.

We will now use the API again to get information about the launches using the IDs given for each launch. Specifically, we will be using columns rocket, payloads, Launchpad, cores.

```
# Lets take a subset of our dataframe keeping only the features we want and the flight number, and date_utc.
data = data[['rocket', 'payloads', 'launchpad', 'cores', 'flight_number', 'date_utc']]

# We will remove rows with multiple cores because those are falcon rockets with 2 extra rocket boosters and rows that have multiple payloads in a single rocket.
data = data[data['cores'].map(len)==1]
data = data[data['payloads'].map(len)==1]

# Since payloads and cores are lists of size 1 we will also extract the single value in the list and replace the feature.
data['cores'] = data['cores'].map(lambda x : x[0])
data['payloads'] = data['payloads'].map(lambda x : x[0])

# We also want to convert the date_utc to a datetime datatype and then extracting the date leaving the time
data['date'] = pd.to_datetime(data['date_utc']).dt.date

# Using the date we will restrict the dates of the launches
data = data[data['date'] <= datetime.date(2020, 11, 13)]
```

* From the rocket we would like to learn the booster name
* From the payload we would like to learn the mass of the payload and the orbit that it is going to
* From the Launchpad we would like to know the name of the launch site being used, the longitude, and the latitude.
* From cores we would like to learn the outcome of the landing, the type of the landing, number of flights with that core, whether gridfins were used, whether the core is reused, whether legs were used, the landing pad

used, the block of the core which is a number used to seperate version of cores, the number of times this specific core has been reused, and the serial of the core.

The data from these requests will be stored in lists and will be used to create a new dataframe.

```
#Global variables
BoosterVersion = []
PayloadMass = []
Orbit = []
LaunchSite = []
Outcome = []
Flights = []
GridFins = []
Reused = []
Legs = []
LandingPad = []
Block = []
ReusedCount = []
Serial = []
Longitude = []
Latitude = []
```

These functions will apply the outputs globally to the above variables. Let's take a looks at BoosterVersion variable. Before we apply getBoosterVersion the list is empty:

```
BoosterVersion
```

→ kết quả: [ ]

Now, let's apply getBoosterVersion function method to get the booster version

```
# Call getBoosterVersion
getBoosterVersion(data)
```

the list has now been update

```
BoosterVersion[0:5]
```

kết quả: ['Falcon 1', 'Falcon 1', 'Falcon 1', 'Falcon 1', 'Falcon 9']

we can apply the rest of the functions here:

```
# Call getLaunchSite
getLaunchSite(data)
# Call getPayloadData
getPayloadData(data)
# Call getCoreData
getCoreData(data)
```

Finally, let’s construct our dataset using the data we have obtained. We we combine the columns into a dictionary.

```
launch_dict = {'FlightNumber': list(data['flight_number']),
'Date': list(data['date']),
'BoosterVersion':BoosterVersion,
'PayloadMass':PayloadMass,
'Orbit':Orbit,
'LaunchSite':LaunchSite,
'Outcome':Outcome,
'Flights':Flights,
'GridFins':GridFins,
'Reused':Reused,
'Legs':Legs,
'LandingPad':LandingPad,
'Block':Block,
'ReusedCount':ReusedCount,
'Serial':Serial,
'Longitude': Longitude,
'Latitude': Latitude}
```

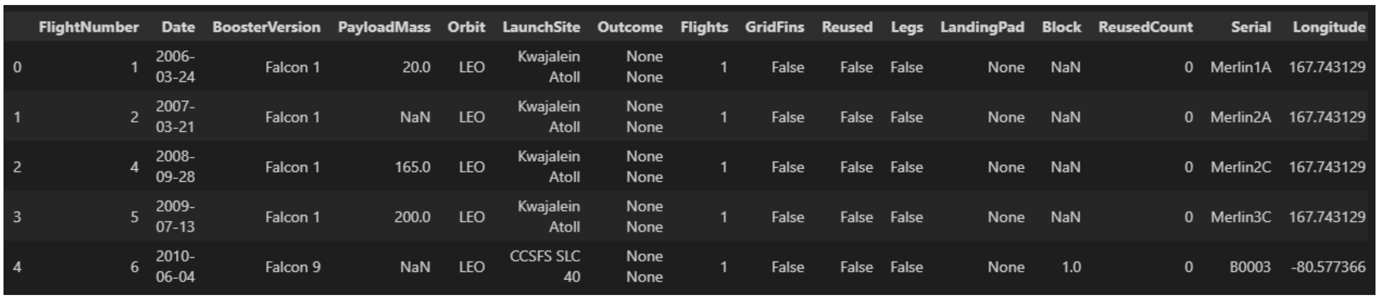
create a Pandas data frame from the dictionary

Then, we need to create a Pandas data frame from the dictionary launch_dict.

```
# Create a data from launch_dict
data = pd.DataFrame(launch_dict)
```

Show the summary of the dataframe

```
# Show the head of the dataframe
data.head()
```

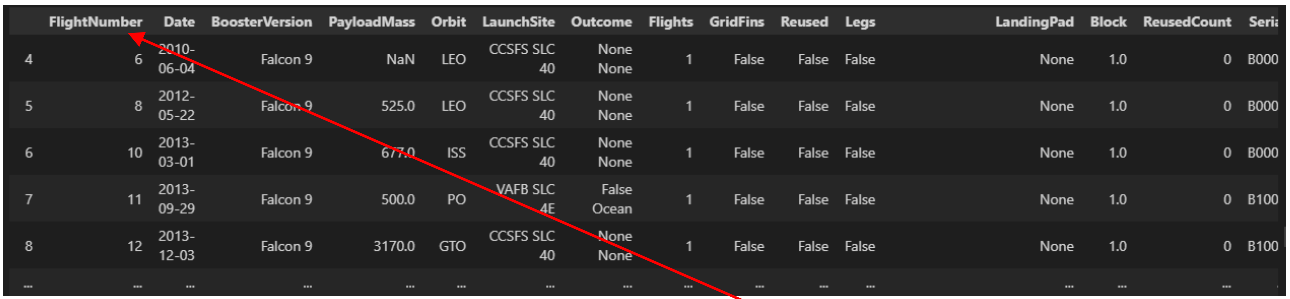


	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPad	Block	ReusedCount	Serial	Longitude
0	1	2006-03-24	Falcon 1	20.0	LEO	Kwajalein Atoll	None None	1	False	False	False	None	NaN	0	Merlin1A	167.743129
1	2	2007-03-21	Falcon 1	NaN	LEO	Kwajalein Atoll	None None	1	False	False	False	None	NaN	0	Merlin2A	167.743129
2	4	2008-09-28	Falcon 1	165.0	LEO	Kwajalein Atoll	None None	1	False	False	False	None	NaN	0	Merlin2C	167.743129
3	5	2009-07-13	Falcon 1	200.0	LEO	Kwajalein Atoll	None None	1	False	False	False	None	NaN	0	Merlin3C	167.743129
4	6	2010-06-04	Falcon 9	NaN	LEO	CCSFS SLC 40	None None	1	False	False	False	None	1.0	0	B0003	-80.577366

Task 2: Filter the dataframe to only include `Falcon 9` launches

Finally, we will remove the Falcon 1 launches keeping only the Falcon 9 launches. Filter the data dataframe using the BoosterVersion column to only keep the Falcon 9 launches. Save the filtered data to a new dataframe called data_falcon9.

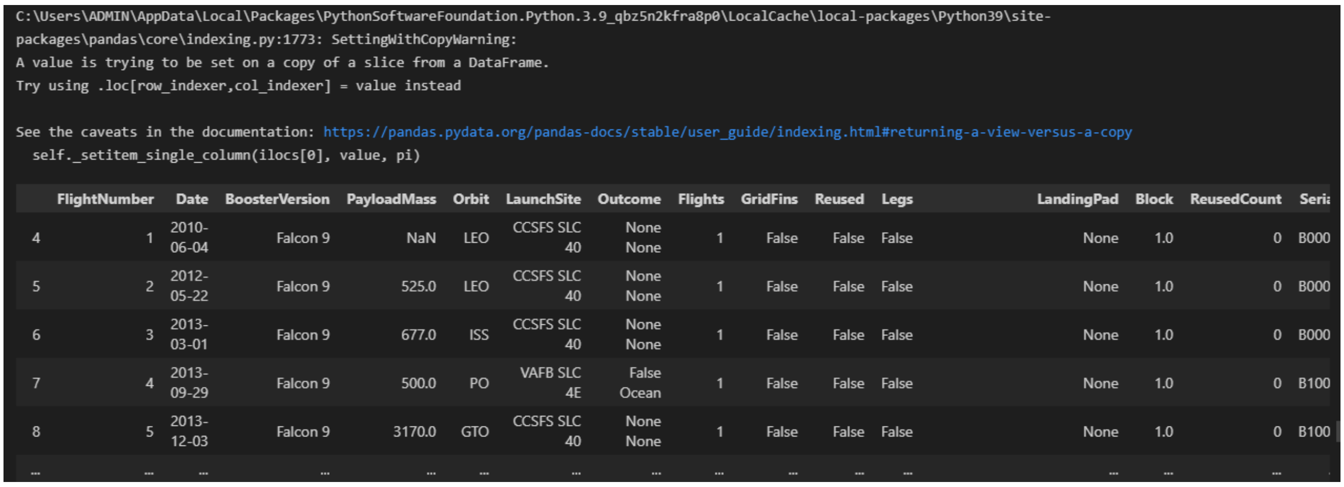
```
# Hint data['BoosterVersion']!='Falcon 1'
data_falcon9 = data[data.BoosterVersion == 'Falcon 9']
data_falcon9
```



	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPad	Block	ReusedCount	Seri
4	6	2010-06-04	Falcon 9	NaN	LEO	CCSFS SLC 40	None None	1	False	False	False	None	1.0	0	B000
5	8	2012-05-22	Falcon 9	525.0	LEO	CCSFS SLC 40	None None	1	False	False	False	None	1.0	0	B000
6	10	2013-03-01	Falcon 9	677.0	ISS	CCSFS SLC 40	None None	1	False	False	False	None	1.0	0	B000
7	11	2013-09-29	Falcon 9	500.0	PO	VAFB SLC 4E	False Ocean	1	False	False	False	None	1.0	0	B100
8	12	2013-12-03	Falcon 9	3170.0	GTO	CCSFS SLC 40	None None	1	False	False	False	None	1.0	0	B100
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	

Now that we have removed some values we should reset the FlgihtNumber column

```
data_falcon9.loc[:,'FlightNumber'] = list(range(1, data_falcon9.shape[0]+1))
data_falcon9
```



```
C:\Users\ADMIN\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.9_qbz5n2kfra8p0\LocalCache\local-packages\Python39\site-
packages\pandas\core\indexing.py:1773: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  self._setitem_single_column(ilocs[0], value, pi)
```

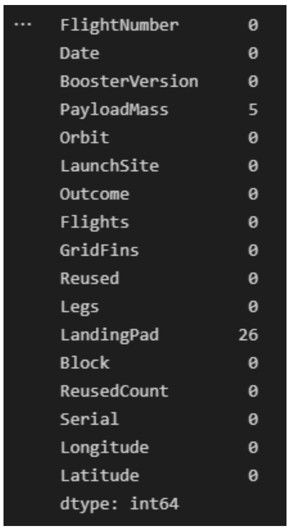
	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPad	Block	ReusedCount	Seri
4	1	2010-06-04	Falcon 9	NaN	LEO	CCSFS SLC 40	None None	1	False	False	False	None	1.0	0	B000
5	2	2012-05-22	Falcon 9	525.0	LEO	CCSFS SLC 40	None None	1	False	False	False	None	1.0	0	B000
6	3	2013-03-01	Falcon 9	677.0	ISS	CCSFS SLC 40	None None	1	False	False	False	None	1.0	0	B000
7	4	2013-09-29	Falcon 9	500.0	PO	VAFB SLC 4E	False Ocean	1	False	False	False	None	1.0	0	B100
8	5	2013-12-03	Falcon 9	3170.0	GTO	CCSFS SLC 40	None None	1	False	False	False	None	1.0	0	B100
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	

Task 3: Dealing with Missing Values

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

finding null values

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

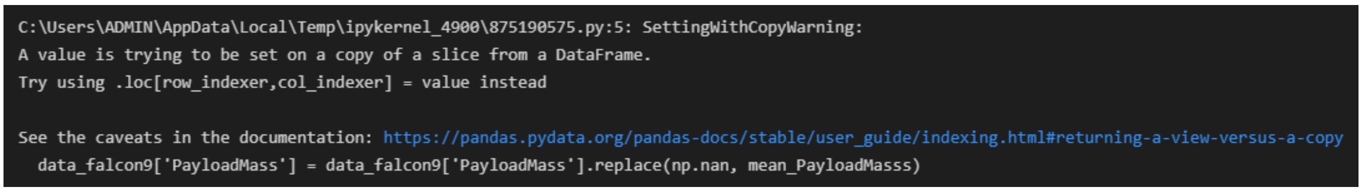


```
...  FlightNumber      0
     Date              0
     BoosterVersion    0
     PayloadMass       5
     Orbit             0
     LaunchSite        0
     Outcome           0
     Flights           0
     GridFins          0
     Reused            0
     Legs              0
     LandingPad       26
     Block             0
     ReusedCount       0
     Serial            0
     Longitude         0
     Latitude          0
     dtype: int64
```

Before we can continue we must deal with these missing values. The LandingPad column will retain None values to represent when landing pads were not used.

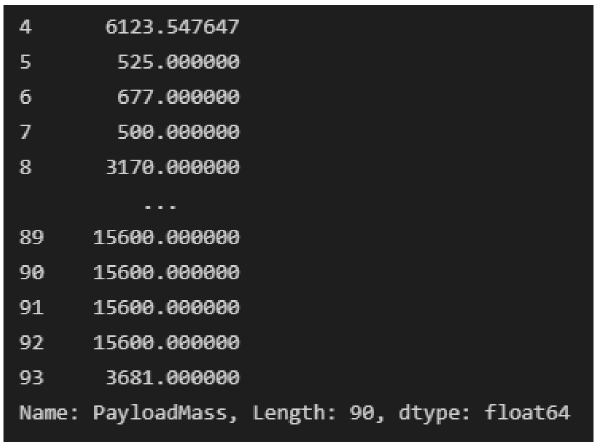
Calculate below the mean for the PayloadMass using the .mean(). Then use the mean and the .replace() function to replace `np.nan` values in the data with the mean you calculated.

```
# Calculate the mean value of PayloadMass column
mean_PayloadMasss = data_falcon9.PayloadMass.mean(axis=0)

# Replace the np.nan values with its mean value
data_falcon9['PayloadMass'] = data_falcon9['PayloadMass'].replace(np.nan, mean_PayloadMasss)
```



```
C:\Users\ADMIN\AppData\Local\Temp\ipykernel_4900\875190575.py:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  data_falcon9['PayloadMass'] = data_falcon9['PayloadMass'].replace(np.nan, mean_PayloadMasss)
```

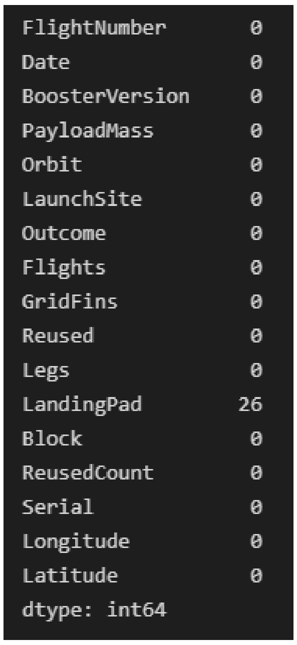
```
data_falcon9['PayloadMass']
```



```
4      6123.547647
5       525.000000
6       677.000000
7       500.000000
8      3170.000000
          ...
89    15600.000000
90    15600.000000
91    15600.000000
92    15600.000000
93     3681.000000
Name: PayloadMass, Length: 90, dtype: float64
```

You should see the number of missing values of the PayLoadMass change to zero.

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



```
FlightNumber       0
Date               0
BoosterVersion     0
PayloadMass        0
Orbit              0
LaunchSite         0
Outcome            0
Flights            0
GridFins           0
Reused             0
Legs               0
LandingPad        26
Block              0
ReusedCount        0
Serial             0
Longitude          0
Latitude           0
dtype: int64
```

Now we should have no missing values in our dataset except for in LandingPad.

save to_csv file (dataset_part_1.csv) for providing data in a pre-selected data range

We can now export it to a CSV for the next section, but to make the answers consistent, in the next lab we will provide data in a pre-selected date range.

```
data_falcon9.to_csv('dataset_part_1.csv', index=False)
```

2/ Data wrangling

In this lab, we will perform some Exploratory Data Analysis (EDA) to find some patterns in the data and determine what would be the label for training supervised models.

In the data set, there are several different cases where the booster did not land successfully. Sometimes a landing was attempted but failed due to an accident; for example, True Ocean means the mission outcome was successfully landed to a specific region of the ocean while False Ocean means the mission outcome was unsuccessfully landed to a specific region of the ocean.

- True RTLS means the mission outcome was successfully landed to a ground pad
- False RTLS means the mission outcome was unsuccessfully landed to a ground pad.
- True ASDS means the mission outcome was successfully landed on a drone ship.
- False ASDS means the mission outcome was unsuccessfully landed on a drone ship.

In this lab we will mainly convert those outcomes into Training Labels with

- `1` means the booster successfully landed
- `0` means it was unsuccessful.

Objectives

Perform exploratory Data Analysis and determine Training Labels

* Exploratory Data Analysis

* Determine Training Labels

Import Libraries and Define Auxiliary Functions

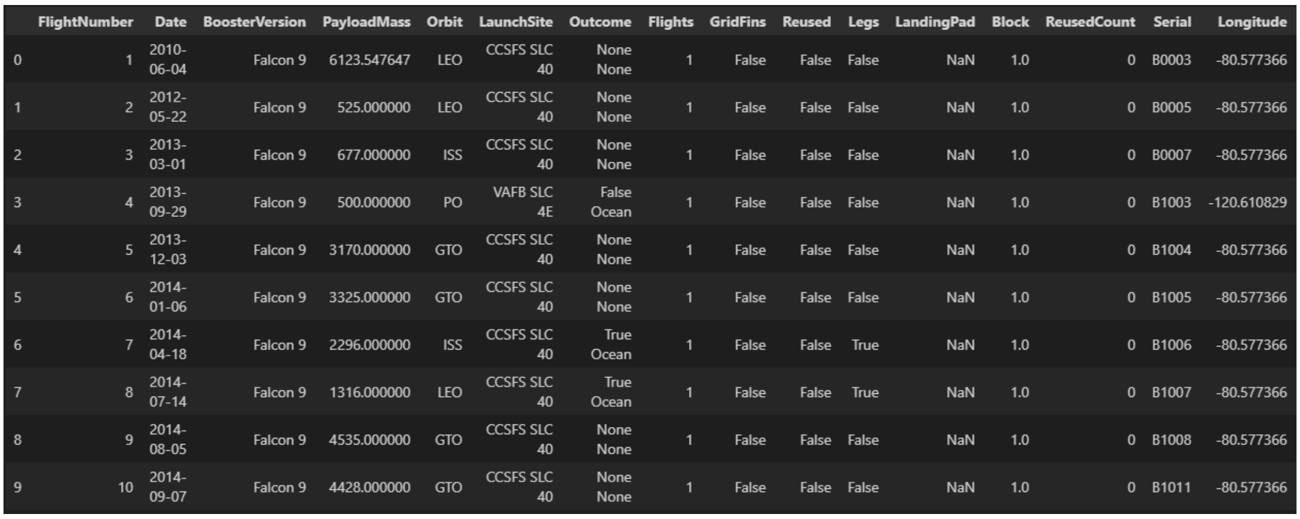
We will import the following libraries.

```
# Pandas is a software library written for the Python programming language for data manipulation and analysis.
import pandas as pd
#NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays
import numpy as np
```

Data Analysis

Load Space X dataset, from last section.

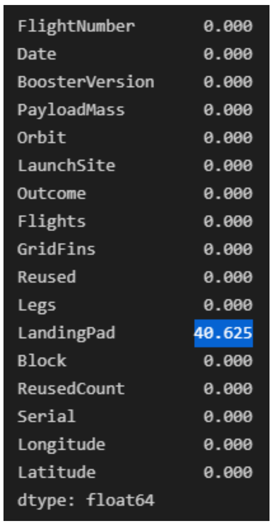
```
#df=pd.read_csv("https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/dataset_part_1.csv")
df=pd.read_csv("dataset_part_1.csv")

df.head(10)
```



	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPad	Block	ReusedCount	Serial	Longitude
0	1	2010-06-04	Falcon 9	6123.547647	LEO	CCSFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B0003	-80.577366
1	2	2012-05-22	Falcon 9	525.000000	LEO	CCSFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B0005	-80.577366
2	3	2013-03-01	Falcon 9	677.000000	ISS	CCSFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B0007	-80.577366
3	4	2013-09-29	Falcon 9	500.000000	PO	VAFB SLC 4E	False Ocean	1	False	False	False	NaN	1.0	0	B1003	-120.610829
4	5	2013-12-03	Falcon 9	3170.000000	GTO	CCSFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B1004	-80.577366
5	6	2014-01-06	Falcon 9	3325.000000	GTO	CCSFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B1005	-80.577366
6	7	2014-04-18	Falcon 9	2296.000000	ISS	CCSFS SLC 40	True Ocean	1	False	False	True	NaN	1.0	0	B1006	-80.577366
7	8	2014-07-14	Falcon 9	1316.000000	LEO	CCSFS SLC 40	True Ocean	1	False	False	True	NaN	1.0	0	B1007	-80.577366
8	9	2014-08-05	Falcon 9	4535.000000	GTO	CCSFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B1008	-80.577366
9	10	2014-09-07	Falcon 9	4428.000000	GTO	CCSFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B1011	-80.577366

Identify and calculate the percentage of the missing values in each attribute

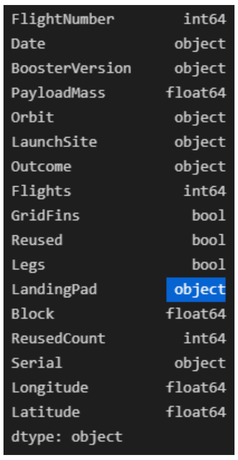
```
df.isnull().sum()/df.count()*100
```



```
FlightNumber      0.000
Date              0.000
BoosterVersion    0.000
PayloadMass       0.000
Orbit             0.000
LaunchSite        0.000
Outcome           0.000
Flights           0.000
GridFins          0.000
Reused            0.000
Legs              0.000
LandingPad       40.625
Block             0.000
ReusedCount       0.000
Serial            0.000
Longitude         0.000
Latitude          0.000
dtype: float64
```

Identify which columns are numerical and categorical:

```
df.dtypes
```



```
FlightNumber        int64
Date               object
BoosterVersion     object
PayloadMass       float64
Orbit              object
LaunchSite         object
Outcome            object
Flights             int64
GridFins             bool
Reused               bool
Legs                 bool
LandingPad         object
Block             float64
ReusedCount         int64
Serial             object
Longitude         float64
Latitude          float64
dtype: object
```

TASK 1: Calculate the number of launches on each site

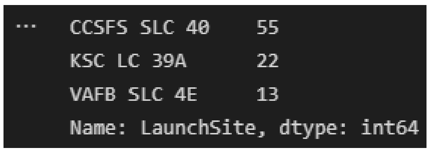
The data contains several Space X launch facilities: Cape Canaveral Space Launch Complex 40 VAFB SLC 4E, Vandenberg Air Force Base Space Launch Complex 4E (SLC-4E), Kennedy Space Center Launch Complex 39A KSC LC 39A. The location of each Launch Is placed in the column LaunchSite.

Next, let's see the number of launches for each site.

determine the number of launches on each site

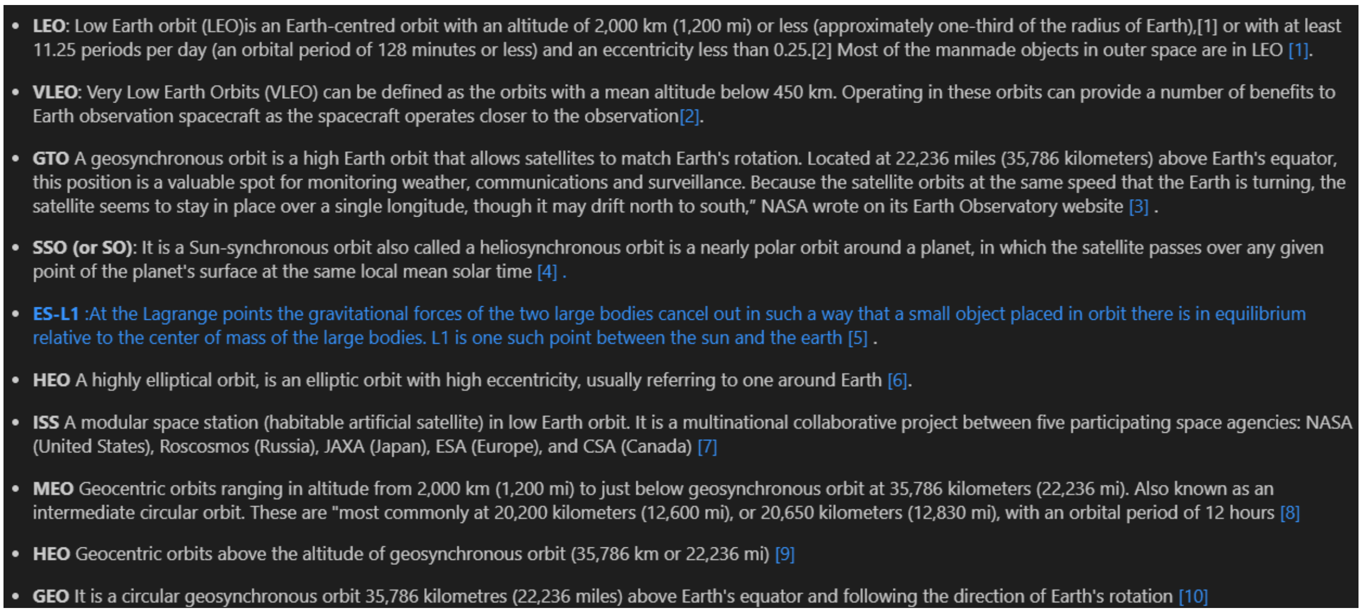
Use the method .value_counts() on the column LaunchSite to determine the number of launches on each site:

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

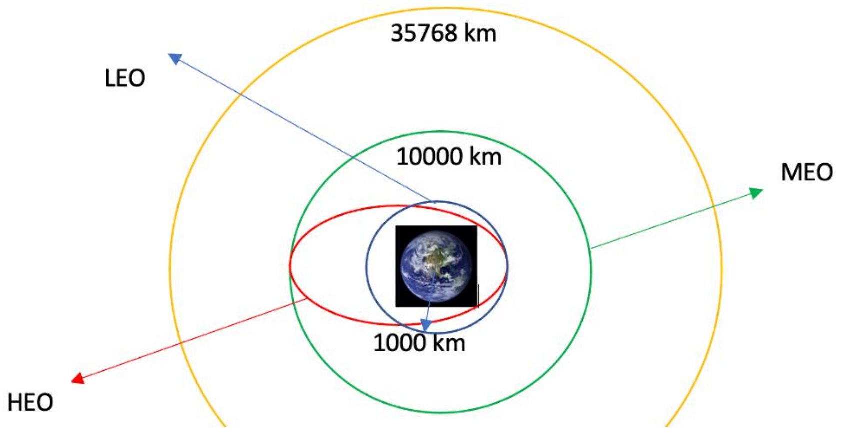


```
···  CCSFS SLC 40    55
     KSC LC 39A      22
     VAFB SLC 4E     13
     Name: LaunchSite, dtype: int64
```

Each launch aims to a dedicated orbit, and here are some common orbit types:



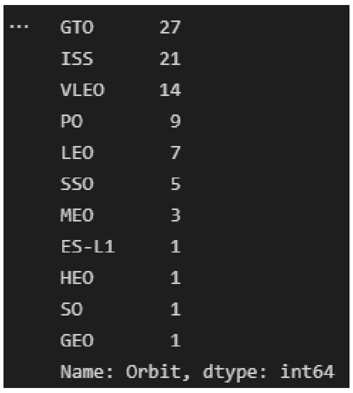
- **LEO**: Low Earth orbit (LEO)is an Earth-centred orbit with an altitude of 2,000 km (1,200 mi) or less (approximately one-third of the radius of Earth),[1] or with at least 11.25 periods per day (an orbital period of 128 minutes or less) and an eccentricity less than 0.25.[2] Most of the manmade objects in outer space are in LEO [1].
- **VLEO**: Very Low Earth Orbits (VLEO) can be defined as the orbits with a mean altitude below 450 km. Operating in these orbits can provide a number of benefits to Earth observation spacecraft as the spacecraft operates closer to the observation[2].
- **GTO** A geosynchronous orbit is a high Earth orbit that allows satellites to match Earth's rotation. Located at 22,236 miles (35,786 kilometers) above Earth's equator, this position is a valuable spot for monitoring weather, communications and surveillance. Because the satellite orbits at the same speed that the Earth is turning, the satellite seems to stay in place over a single longitude, though it may drift north to south," NASA wrote on its Earth Observatory website [3] .
- **SSO (or SO)**: It is a Sun-synchronous orbit also called a heliosynchronous orbit is a nearly polar orbit around a planet, in which the satellite passes over any given point of the planet's surface at the same local mean solar time [4] .
- **ES-L1** :At the Lagrange points the gravitational forces of the two large bodies cancel out in such a way that a small object placed in orbit there is in equilibrium relative to the center of mass of the large bodies. L1 is one such point between the sun and the earth [5] .
- **HEO** A highly elliptical orbit, is an elliptic orbit with high eccentricity, usually referring to one around Earth [6].
- **ISS** A modular space station (habitable artificial satellite) in low Earth orbit. It is a multinational collaborative project between five participating space agencies: NASA (United States), Roscosmos (Russia), JAXA (Japan), ESA (Europe), and CSA (Canada) [7]
- **MEO** Geocentric orbits ranging in altitude from 2,000 km (1,200 mi) to just below geosynchronous orbit at 35,786 kilometers (22,236 mi). Also known as an intermediate circular orbit. These are "most commonly at 20,200 kilometers (12,600 mi), or 20,650 kilometers (12,830 mi), with an orbital period of 12 hours [8]
- **HEO** Geocentric orbits above the altitude of geosynchronous orbit (35,786 km or 22,236 mi) [9]
- **GEO** It is a circular geosynchronous orbit 35,786 kilometres (22,236 miles) above Earth's equator and following the direction of Earth's rotation [10]



TASK 2: Calculate the number and occurrence of each orbit

Use the method .value_counts() to determine the number and occurrence of each orbit in the column Orbit.

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

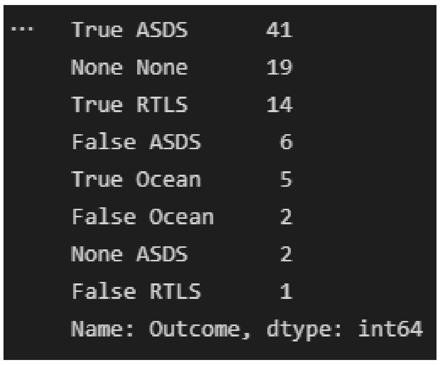


```
···  GTO      27
     ISS      21
     VLEO     14
     PO        9
     LEO       7
     SSO       5
     MEO       3
     ES-L1     1
     HEO       1
     SO        1
     GEO       1
     Name: Orbit, dtype: int64
```

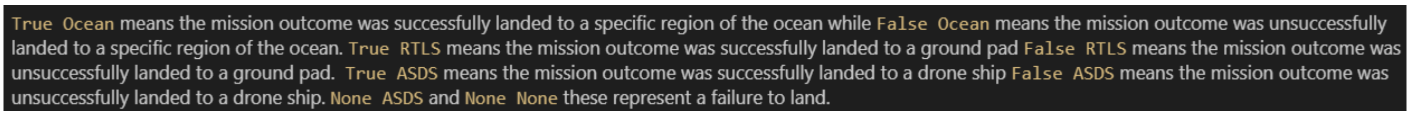
TASK 3: Calculate the number and occurrence of mission outcome per orbit type

Use the method .value_counts() on the column Outcome to determine the number of landing_outcomes. Then assign it to a variable landing_outcomes.

```
# landing_outcomes = values on Outcome column
landing_outcomes = df['Outcome'].value_counts()

landing_outcomes
```

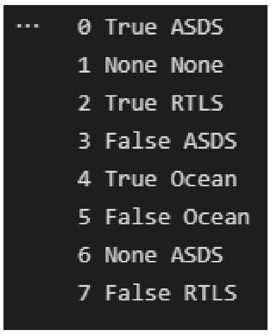


```
···  True ASDS      41
     None None      19
     True RTLS      14
     False ASDS      6
     True Ocean      5
     False Ocean     2
     None ASDS       2
     False RTLS      1
     Name: Outcome, dtype: int64
```



`True Ocean` means the mission outcome was successfully landed to a specific region of the ocean while `False Ocean` means the mission outcome was unsuccessfully landed to a specific region of the ocean. `True RTLS` means the mission outcome was successfully landed to a ground pad `False RTLS` means the mission outcome was unsuccessfully landed to a ground pad. `True ASDS` means the mission outcome was successfully landed to a drone ship `False ASDS` means the mission outcome was unsuccessfully landed to a drone ship. `None ASDS` and `None None` these represent a failure to land.

```
for i,outcome in enumerate(landing_outcomes.keys()):
    print(i,outcome)
```



```
···  0 True ASDS
     1 None None
     2 True RTLS
     3 False ASDS
     4 True Ocean
     5 False Ocean
     6 None ASDS
     7 False RTLS
```

We create a set of outcomes where **the second stage** did not land successfully:

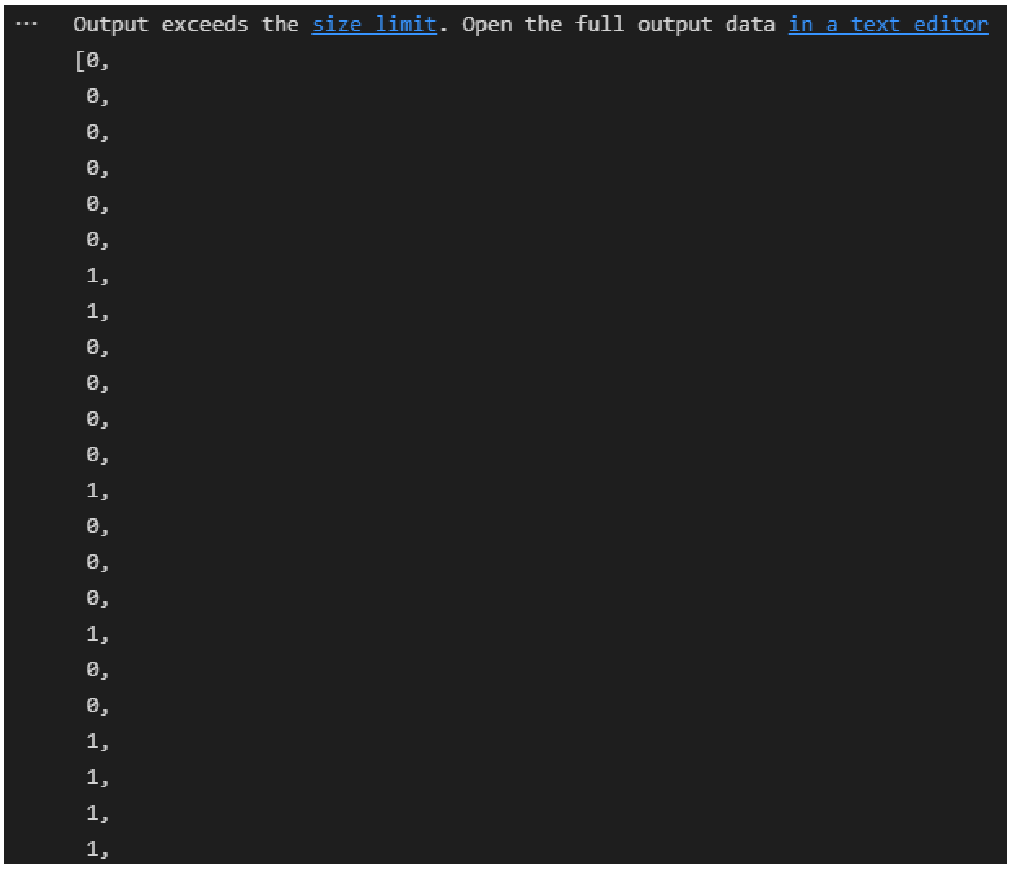
```
bad_outcomes=set(landing_outcomes.keys()[[1,3,5,6,7]])
bad_outcomes
```

kết quả: {'False ASDS', 'False Ocean', 'False RTLS', 'None ASDS', 'None None'}

TASK 4: Create a landing outcome label from Outcome column

Using the Outcome, create a list where the element is zero (0) if the corresponding row in Outcome is in the set bad_outcome; otherwise, it's one (1). Then assign it to the variable landing_class:

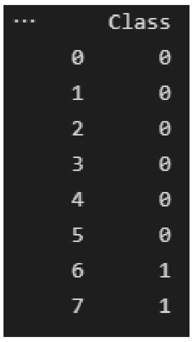
```
# landing_class = 0 if bad_outcome
# landing_class = 1 otherwise
landing_class = []
for outcome in df['Outcome']:
   if outcome in bad_outcomes:
      landing_class.append(0)
   else:
      landing_class.append(1)
landing_class
```



```
···   Output exceeds the size limit. Open the full output data in a text editor
      [0,
       0,
       0,
       0,
       0,
       0,
       1,
       1,
       0,
       0,
       0,
       0,
       1,
       0,
       0,
       0,
       1,
       0,
       0,
       1,
       1,
       1,
       1,
```

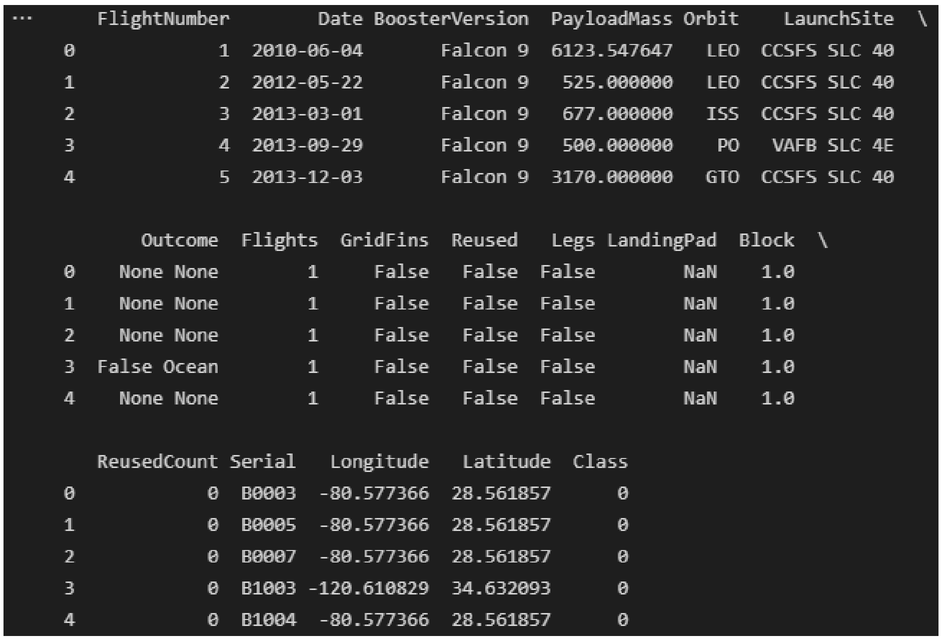
This variable will represent the classification variable that represents the outcome of each launch. If the value is zero, the first stage did not land successfully; one means the first stage landed successfully.

```
df['Class']=landing_class
df[['Class']].head(8)
```



```
···      Class
     0       0
     1       0
     2       0
     3       0
     4       0
     5       0
     6       1
     7       1
```

```
df.head(5)
```



```
…     FlightNumber        Date BoosterVersion  PayloadMass Orbit    LaunchSite  \
0                1  2010-06-04       Falcon 9  6123.547647   LEO  CCSFS SLC 40
1                2  2012-05-22       Falcon 9   525.000000   LEO  CCSFS SLC 40
2                3  2013-03-01       Falcon 9   677.000000   ISS  CCSFS SLC 40
3                4  2013-09-29       Falcon 9   500.000000    PO   VAFB SLC 4E
4                5  2013-12-03       Falcon 9  3170.000000   GTO  CCSFS SLC 40

       Outcome  Flights  GridFins  Reused   Legs LandingPad  Block  \
0    None None        1     False   False  False        NaN    1.0
1    None None        1     False   False  False        NaN    1.0
2    None None        1     False   False  False        NaN    1.0
3  False Ocean        1     False   False  False        NaN    1.0
4    None None        1     False   False  False        NaN    1.0

   ReusedCount Serial   Longitude   Latitude  Class
0            0  B0003  -80.577366  28.561857      0
1            0  B0005  -80.577366  28.561857      0
2            0  B0007  -80.577366  28.561857      0
3            0  B1003 -120.610829  34.632093      0
4            0  B1004  -80.577366  28.561857      0
```

We can use the following line of code to determine the success rate:

```
df["Class"].mean()
```

kết quả: 0.6666666666666666

save dataframe to_csv ("dataset_part_2.csv")

We can now export it to a CSV for the next section, but to make the answers consistent, in the next lab we will provide data in a pre-selected date range.

```
df.to_csv("dataset_part_2.csv", index=False)
```

3/ Exploring and Preparing Data (EDA) and Preparing Data Feature Engineering

In this assignment, we will predict if the Falcon 9 first stage will land successfully. SpaceX advertises Falcon 9 rocket launches on its website with a cost of 62 million dollars; other providers cost upward of 165 million dollars each, much of the savings is due to the fact that SpaceX can reuse the first stage.

In this lab, you will perform Exploratory Data Analysis and Feature Engineering.

Objectives

Perform exploratory Data Analysis and Feature Engineering using `Pandas` and `Matplotlib`

* Exploratory Data Analysis

* Preparing Data Feature Engineering

Import Libraries and Define Auxiliary Functions

We will import the following libraries the lab:

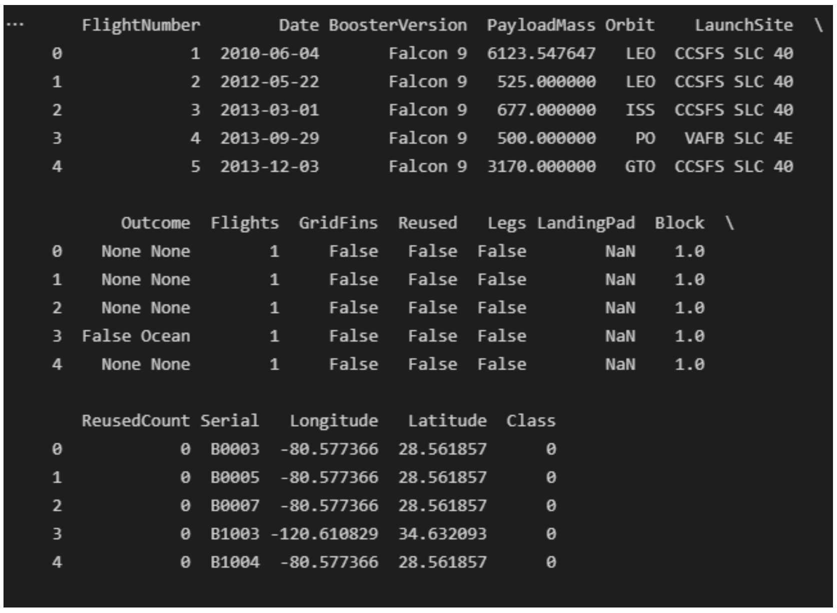
```
# pandas is a software library written for the Python programming language for data manipulation and analysis.
import pandas as pd

#NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays
import numpy as np

# Matplotlib is a plotting library for python and pyplot gives us a MatLab like plotting framework. We will use this in our plotter function to plot data.
import matplotlib.pyplot as plt

#Seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics
import seaborn as sns
```

Exploratory Data Analysis (EDA)

pd.read_csv a dataset into a Pandas dataframe

First, let's read the SpaceX dataset into a Pandas dataframe and print its summary

```
# df=pd.read_csv("https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/dataset_part_2.csv")

# If you were unable to complete the previous lab correctly you can uncomment and load this csv

# df = pd.read_csv('https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-DS0701EN-SkillsNetwork/api/dataset_part_2.csv')

df=pd.read_csv("dataset_part_2.csv")

df.head(5)
```

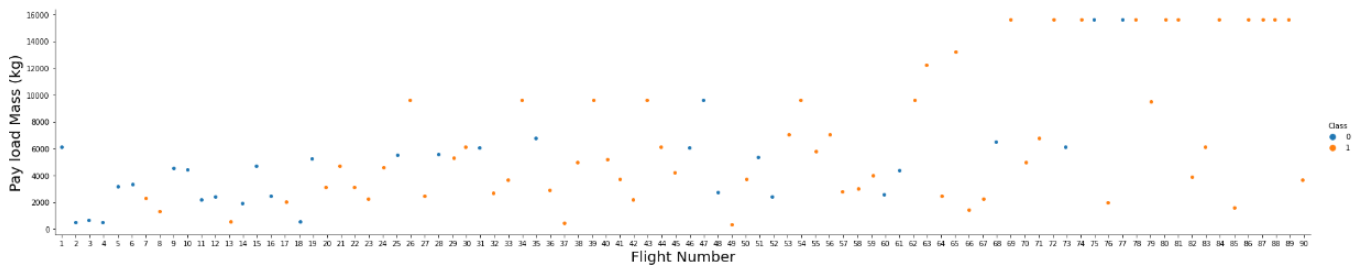


```
...   FlightNumber        Date BoosterVersion  PayloadMass Orbit    LaunchSite  \
   0             1  2010-06-04       Falcon 9  6123.547647   LEO  CCSFS SLC 40
   1             2  2012-05-22       Falcon 9   525.000000   LEO  CCSFS SLC 40
   2             3  2013-03-01       Falcon 9   677.000000   ISS  CCSFS SLC 40
   3             4  2013-09-29       Falcon 9   500.000000    PO   VAFB SLC 4E
   4             5  2013-12-03       Falcon 9  3170.000000   GTO  CCSFS SLC 40

         Outcome  Flights  GridFins  Reused   Legs LandingPad  Block  \
   0   None None        1     False   False  False        NaN    1.0
   1   None None        1     False   False  False        NaN    1.0
   2   None None        1     False   False  False        NaN    1.0
   3 False Ocean        1     False   False  False        NaN    1.0
   4   None None        1     False   False  False        NaN    1.0

     ReusedCount Serial   Longitude   Latitude  Class
   0           0  B0003  -80.577366  28.561857      0
   1           0  B0005  -80.577366  28.561857      0
   2           0  B0007  -80.577366  28.561857      0
   3           0  B1003 -120.610829  34.632093      0
   4           0  B1004  -80.577366  28.561857      0
```

First, let's try to see how the `FlightNumber` (indicating the continuous launch attempts.) and `Payload` variables would affect the launch outcome.

We can plot out the FlightNumber vs. PayloadMass and overlay the outcome of the launch. We see that as the flight number increases, the first stage is more likely to land successfully. The payload mass is also important; it seems the more massive the payload, the less likely the first stage will return.

```
sns.catplot(y="PayloadMass", x="FlightNumber", hue="Class", data=df, aspect = 5)
plt.xlabel("Flight Number",fontsize=20)
plt.ylabel("Pay load Mass (kg)",fontsize=20)
plt.show()
```



We see that different launch sites have different success rates. CCAFS LC-40, has a success rate of 60 %, while KSC LC-39A and VAFB SLC 4E has a success rate of 77%.

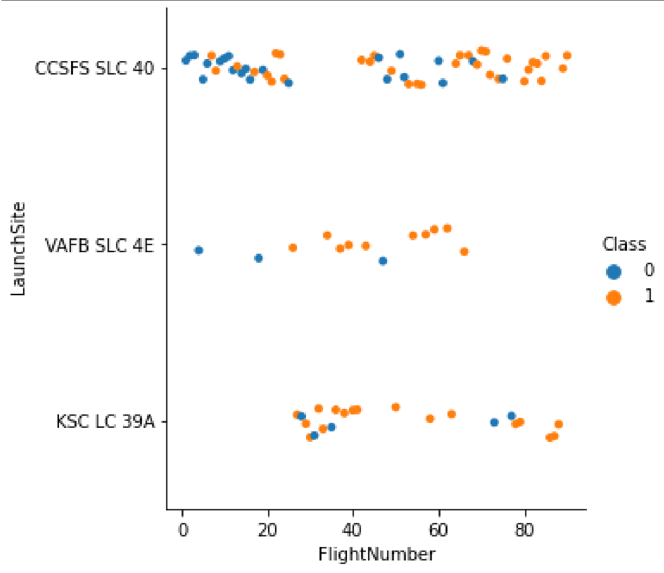
Next, let's drill down to each site visualize its detailed launch records.

TASK 1: Visualize the relationship between Flight Number and Launch Site

Use the function catplot to plot FlightNumber vs LaunchSite,

- set the parameter x parameter to FlightNumber,
- set the y to LaunchSite,
- and set the parameter hue to 'class'

```
# Plot a scatter point chart with x axis to be Flight Number and y axis to be the launch site, and hue to be the class value
sns.catplot(y="LaunchSite", x="FlightNumber", hue="Class", data=df)
```

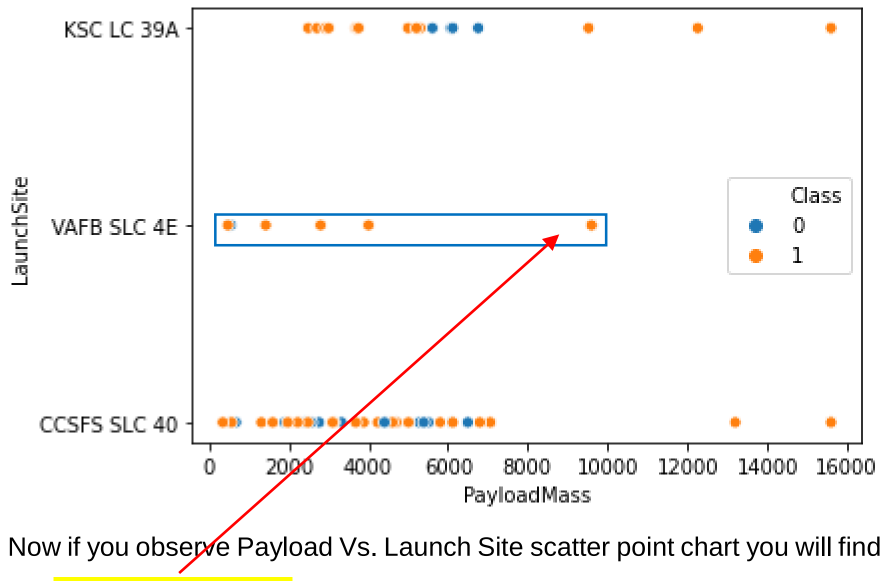


Now try to explain the patterns you found in the Flight Number vs. Launch Site scatter point plots.

TASK 2: Visualize the relationship between Payload and Launch Site

We also want to observe if there is any relationship between launch sites and their payload mass.

```
# Plot a scatter point chart with x axis to be Pay Load Mass (kg) and y axis to be the launch site, and hue to be the class value
sns.scatterplot(x='PayloadMass', y='LaunchSite', hue='Class', data=df)
```



Now if you observe Payload Vs. Launch Site scatter point chart you will find for the VAFB-SLC launchsite there are no rockets launched for heavy PayloadMass (greater than 10000).

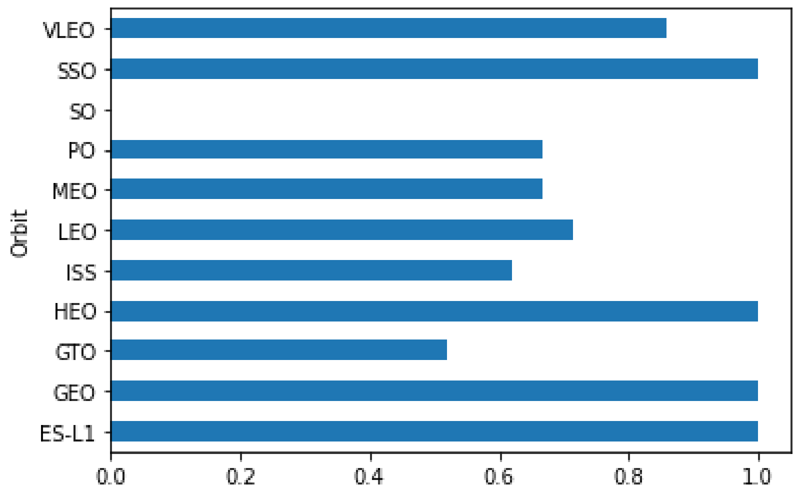
TASK 3: Visualize the relationship between success rate of each orbit type

Next, we want to visually check if there are any relationship between success rate and orbit type.

create bar chart

Let's create a `bar chart` for the successful rate of each orbit

```
# HINT use groupby method on Orbit column and get the mean of Class column
success_rates = df.groupby('Orbit')['Class'].mean()
success_rates.plot.barh()
```



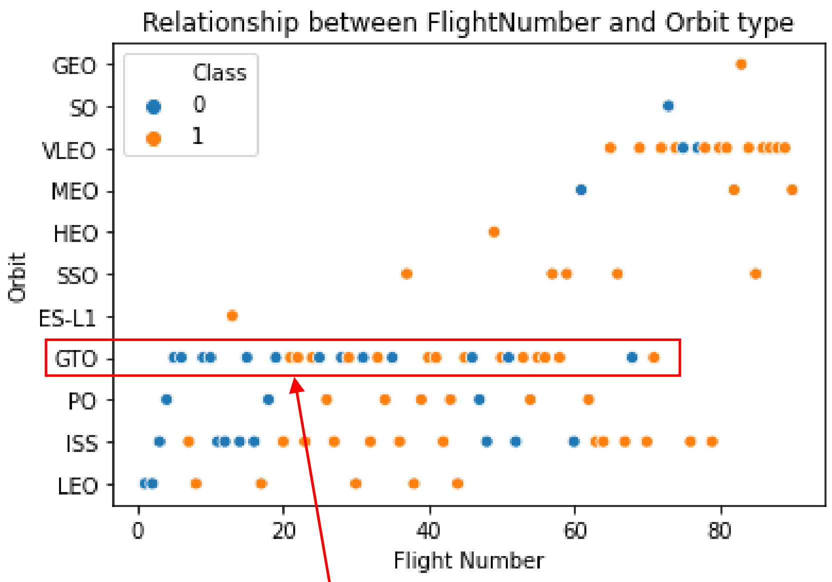
Analyze the plotted bar chart try to find which orbits have high success rate.

-->SSO, HEO, GEO, ES-L1

TASK 4: Visualize the relationship between FlightNumber and Orbit type

For each orbit, we want to see if there is any relationship between FlightNumber and Orbit type.

```
# Plot a scatter point chart with x axis to be 'FlightNumber',
# and y axis to be the 'Orbit',
# and hue to be the 'Class' value
sns.scatterplot(x='FlightNumber', y='Orbit', data=df, hue='Class')
plt.title('Relationship between FlightNumber and Orbit type')
plt.xlabel('Flight Number')
plt.ylabel('Orbit')
plt.show()
```

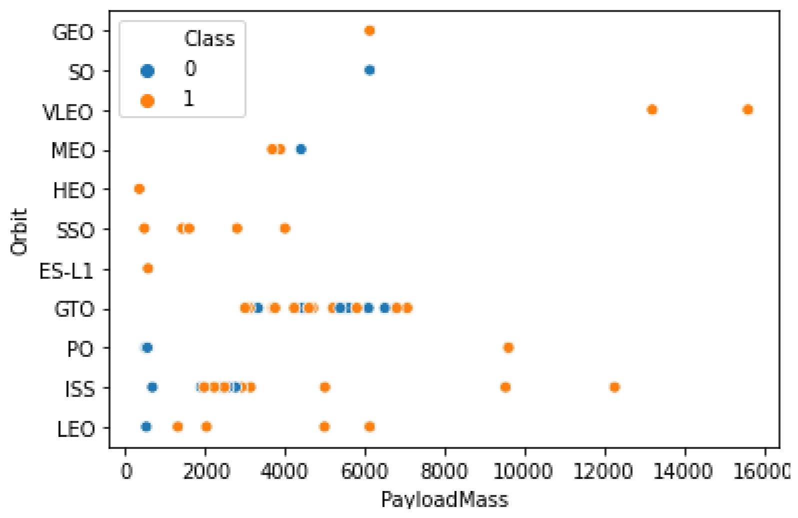


You should see that in the LEO orbit the Success appears related to the number of flights; on the other hand, there seems to be no relationship between flight number when in GTO orbit.

TASK 5: Visualize the relationship between Payload and Orbit type

Similarly, we can plot the Payload vs. Orbit scatter point charts to reveal the relationship between Payload and Orbit type

```
# Plot a scatter point chart with x axis to be Payload and y axis to be the Orbit, and hue to be the class value
sns.scatterplot(x='PayloadMass', y='Orbit', data=df, hue='Class')
```



With heavy payloads the successful landing or positive landing rate are more for PO, LEO and ISS.

However, for GTO we cannot distinguish this well as both positive landing rate and negative landing (unsuccessful mission) are both there here.

TASK 6: Visualize the launch success yearly trend

You can plot a line chart with x axis to be Year,

and y axis to be average success rate, to get the average launch success trend.

The function will help you get the year from the date:

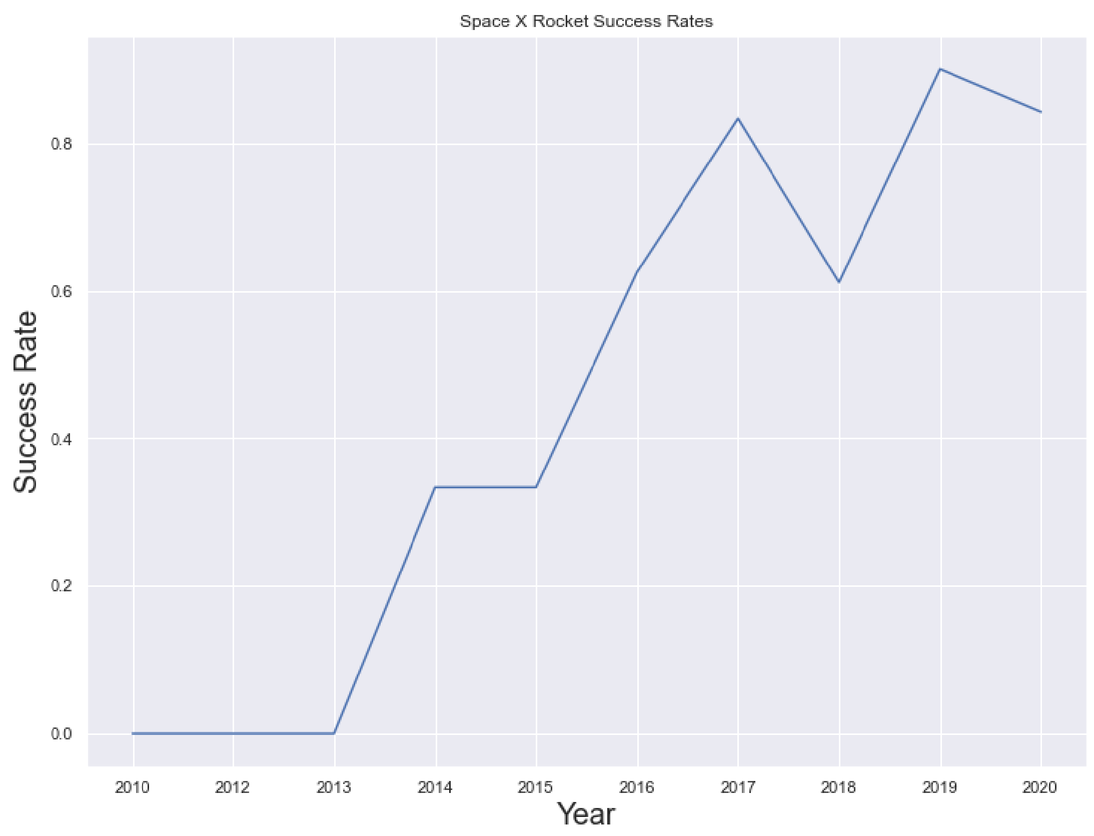
```
# A function to Extract years from the date
year=[]
def Extract_year(date):
    for i in df["Date"]:
        year.append(i.split("-")[0])
    return year
```

```
df['year']=Extract_year(df["Date"])
```

```
df_groupby_year=df.groupby("year",as_index=False)["Class"].mean()
```

Plot a line chart

```
# Plot a line chart with x axis to be the extracted year,
# and y axis to be the success rate
sns.set(rc={'figure.figsize':(12,9)})
sns.lineplot(data=df_groupby_year, x="year", y="Class" )
plt.xlabel("Year",fontsize=20)
plt.title('Space X Rocket Success Rates')
plt.ylabel("Success Rate",fontsize=20)
plt.show()
```



you can observe that the sucess rate since 2013 kept increasing till 2020

save above figure as .png

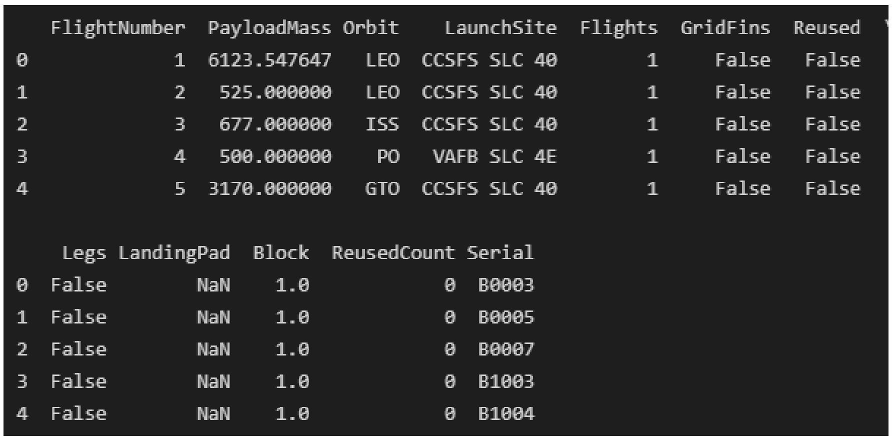
```
plt.savefig('SpaceX Rocket Success Rates.png')
```

Features Engineering

By now, you should obtain some preliminary insights about how each important variable would affect the success rate, we will select the features that will be used in success prediction in the future module.

extract columns features

```
features = df[['FlightNumber', 'PayloadMass', 'Orbit', 'LaunchSite', 'Flights', 'GridFins', 'Reused', 'Legs',
'LandingPad', 'Block', 'ReusedCount', 'Serial']]
features.head()
```



```
   FlightNumber  PayloadMass Orbit    LaunchSite  Flights  GridFins  Reused
0             1  6123.547647   LEO  CCSFS SLC 40        1     False   False
1             2   525.000000   LEO  CCSFS SLC 40        1     False   False
2             3   677.000000   ISS  CCSFS SLC 40        1     False   False
3             4   500.000000    PO   VAFB SLC 4E        1     False   False
4             5  3170.000000   GTO  CCSFS SLC 40        1     False   False

    Legs LandingPad  Block  ReusedCount Serial
0  False        NaN    1.0            0  B0003
1  False        NaN    1.0            0  B0005
2  False        NaN    1.0            0  B0007
3  False        NaN    1.0            0  B1003
4  False        NaN    1.0            0  B1004
```

TASK 7: Create dummy variables to categorical columns

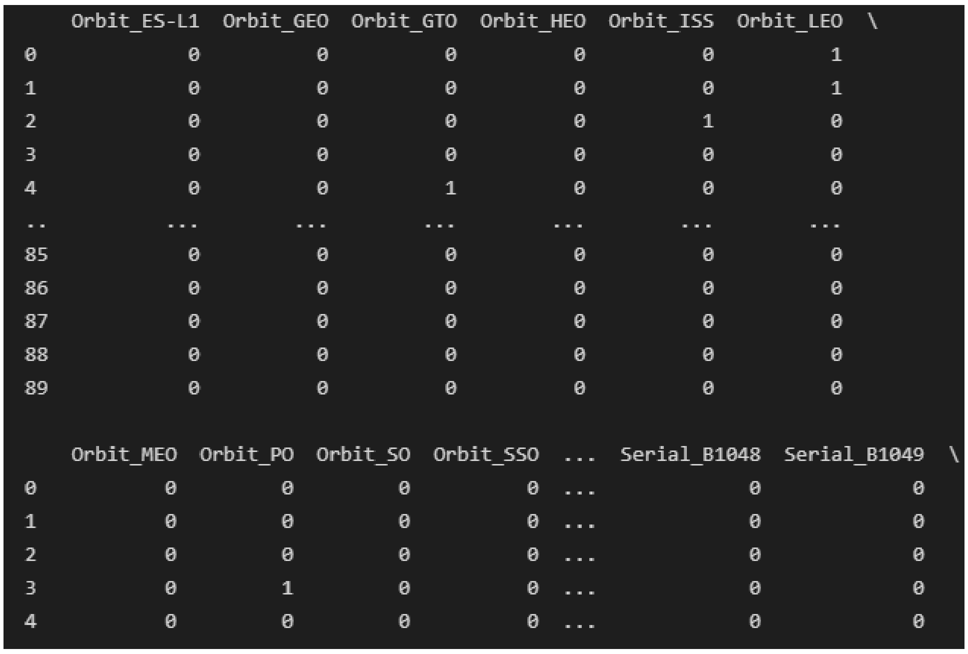
Use the function get_dummies and features dataframe to apply OneHotEncoder to the column Orbits, LaunchSite, LandingPad, and Serial.

pd.get_dummies→ OneHotEncoder

Assign the value to the variable features_one_hot, display the results using the method head. Your result dataframe must include all features including the encoded ones.

```
# HINT: Use get_dummies() function on the categorical columns
features_one_hot = pd.get_dummies(df[['Orbit', 'LaunchSite', 'LandingPad', 'Serial']])
```

```
features_one_hot
```

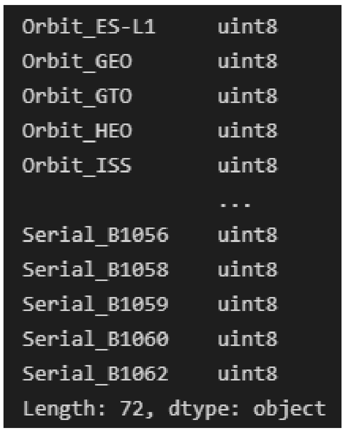


```
    Orbit_ES-L1  Orbit_GEO  Orbit_GTO  Orbit_HEO  Orbit_ISS  Orbit_LEO  \
0             0          0          0          0          0          1
1             0          0          0          0          0          1
2             0          0          0          0          1          0
3             0          0          0          0          0          0
4             0          0          1          0          0          0
..          ...        ...        ...        ...        ...        ...
85            0          0          0          0          0          0
86            0          0          0          0          0          0
87            0          0          0          0          0          0
88            0          0          0          0          0          0
89            0          0          0          0          0          0

    Orbit_MEO  Orbit_PO  Orbit_SO  Orbit_SSO  ...  Serial_B1048  Serial_B1049  \
0           0         0         0          0  ...             0             0
1           0         0         0          0  ...             0             0
2           0         0         0          0  ...             0             0
3           0         1         0          0  ...             0             0
4           0         0         0          0  ...             0             0
```

TASK 8: Cast all numeric columns to `float64`

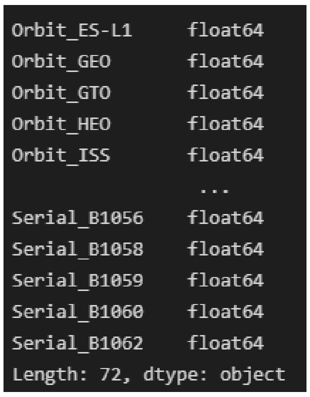
Now that our features_one_hot dataframe only contains numbers cast the entire dataframe to variable type float64.

```
features_one_hot.dtypes
```



```
Orbit_ES-L1      uint8
Orbit_GEO        uint8
Orbit_GTO        uint8
Orbit_HEO        uint8
Orbit_ISS        uint8
                 ...
Serial_B1056     uint8
Serial_B1058     uint8
Serial_B1059     uint8
Serial_B1060     uint8
Serial_B1062     uint8
Length: 72, dtype: object
```

```
# HINT: use astype function
features_one_hot = features_one_hot.astype('float64')
features_one_hot.dtypes
```



```
Orbit_ES-L1      float64
Orbit_GEO        float64
Orbit_GTO        float64
Orbit_HEO        float64
Orbit_ISS        float64
                   ...
Serial_B1056     float64
Serial_B1058     float64
Serial_B1059     float64
Serial_B1060     float64
Serial_B1062     float64
Length: 72, dtype: object
```

We can now export it to a CSV for the next section, but to make the answers consistent, in the next lab we will provide data in a pre-selected date range.

```
features_one_hot.to_csv('dataset_part_3.csv', index=False)
```

4/ Machine Learning Prediction (classification)

SpaceX advertises Falcon 9 rocket launches on its website with a cost of 62 million dollars; other providers cost upward of 165 million dollars each, much of the savings is because SpaceX can reuse the first stage. Therefore, if we can determine if the first stage will land, we can determine the cost of a launch. This information can be used if an alternate company wants to bid against SpaceX for a rocket launch. In this lab, you will create a machine learning pipeline to predict if the first stage will land given the data from the preceding labs.

Objectives

Perform Exploratory Data Analysis and determine Training Labels

* create a column for the class
* standardize the data
* split into training data and test data

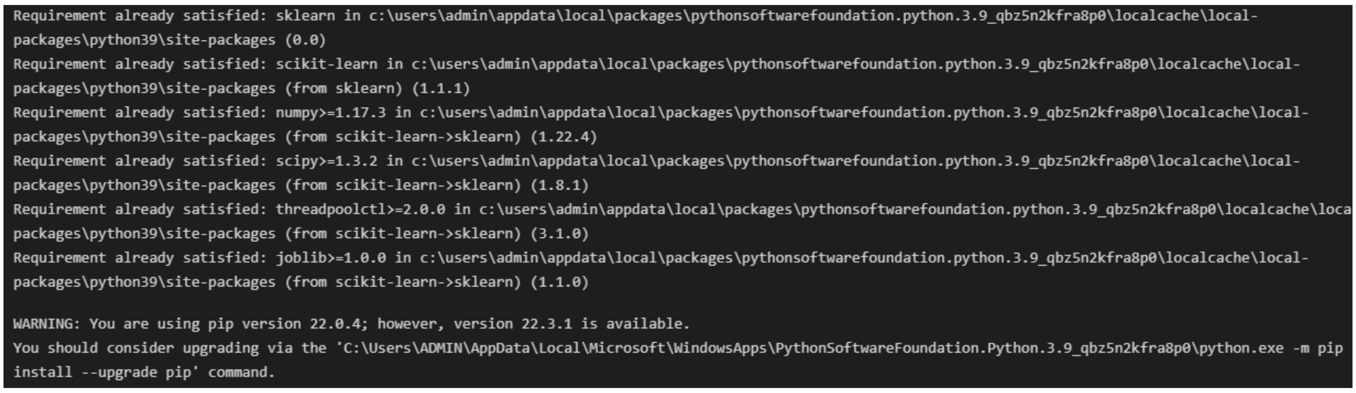
Find best Hyperparameter for SVM, Classification Trees and Logistic Regression

* find the method performs best using test data

Import Libraries and Define Auxiliary Functions

We will import the following libraries for the lab

```
!pip install sklearn
```



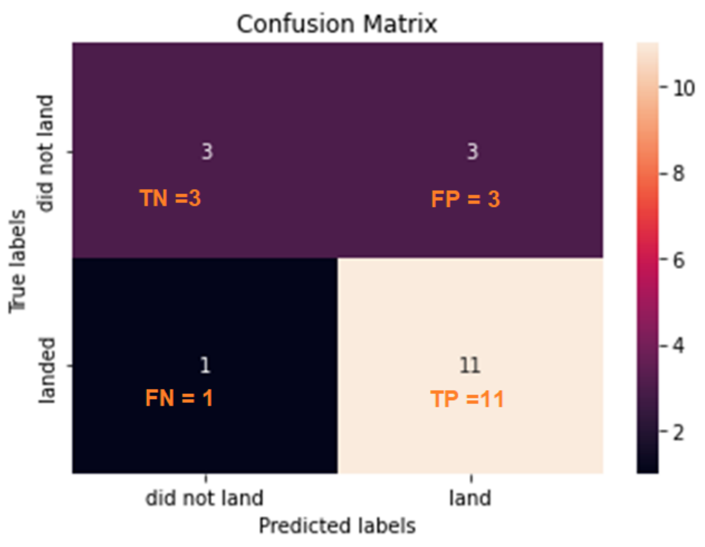
```
Requirement already satisfied: sklearn in c:\users\admin\appdata\local\packages\pythonsoftwarefoundation.python.3.9_qbz5n2kfra8p0\localcache\local-
packages\python39\site-packages (0.0)
Requirement already satisfied: scikit-learn in c:\users\admin\appdata\local\packages\pythonsoftwarefoundation.python.3.9_qbz5n2kfra8p0\localcache\local-
packages\python39\site-packages (from sklearn) (1.1.1)
Requirement already satisfied: numpy>=1.17.3 in c:\users\admin\appdata\local\packages\pythonsoftwarefoundation.python.3.9_qbz5n2kfra8p0\localcache\local-
packages\python39\site-packages (from scikit-learn->sklearn) (1.22.4)
Requirement already satisfied: scipy>=1.3.2 in c:\users\admin\appdata\local\packages\pythonsoftwarefoundation.python.3.9_qbz5n2kfra8p0\localcache\local-
packages\python39\site-packages (from scikit-learn->sklearn) (1.8.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\admin\appdata\local\packages\pythonsoftwarefoundation.python.3.9_qbz5n2kfra8p0\localcache\loca
packages\python39\site-packages (from scikit-learn->sklearn) (3.1.0)
Requirement already satisfied: joblib>=1.0.0 in c:\users\admin\appdata\local\packages\pythonsoftwarefoundation.python.3.9_qbz5n2kfra8p0\localcache\local-
packages\python39\site-packages (from scikit-learn->sklearn) (1.1.0)

WARNING: You are using pip version 22.0.4; however, version 22.3.1 is available.
You should consider upgrading via the 'C:\Users\ADMIN\AppData\Local\Microsoft\WindowsApps\PythonSoftwareFoundation.Python.3.9_qbz5n2kfra8p0\python.exe -m pip
install --upgrade pip' command.
```

```
# Pandas is a software library written for the Python programming language for data manipulation and
analysis.
```

```
import pandas as pd
# NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays
import numpy as np
# Matplotlib is a plotting library for python and pyplot gives us a MatLab like plotting framework. We will use this in our plotter function to plot data.
import matplotlib.pyplot as plt

#Seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics
import seaborn as sns

#----------------------------------------sklearn---------------------------------
# Preprocessing allows us to standardize our data
from sklearn import preprocessing
# Allows us to split our data into training and testing data
from sklearn.model_selection import train_test_split
# Allows us to test parameters of classification algorithms and find the best one
from sklearn.model_selection import GridSearchCV
# ----------------------------------------------------------
# Logistic Regression classification algorithm
from sklearn.linear_model import LogisticRegression
# Support Vector Machine classification algorithm
from sklearn.svm import SVC
# Decision Tree classification algorithm
from sklearn.tree import DecisionTreeClassifier
# K Nearest Neighbors classification algorithm
from sklearn.neighbors import KNeighborsClassifier
```

This function is to plot the confusion matrix.

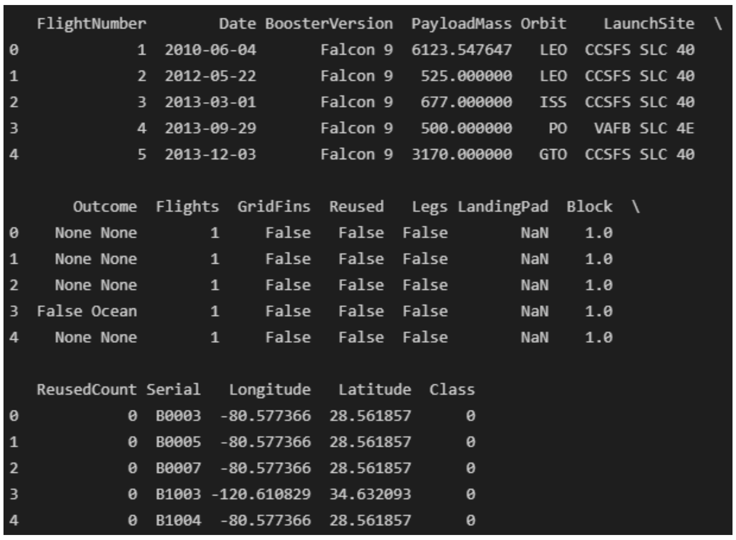
```
def plot_confusion_matrix(y,y_predict):
    "this function plots the confusion matrix"
    from sklearn.metrics import confusion_matrix

    cm = confusion_matrix(y, y_predict)
    ax= plt.subplot()
    sns.heatmap(cm, annot=True, ax = ax);
    #annot=True to annotate cells
    ax.set_xlabel('Predicted labels')
    ax.set_ylabel('True labels')
    ax.set_title('Confusion Matrix');
    ax.xaxis.set_ticklabels(['did not land', 'land']);
    ax.yaxis.set_ticklabels(['did not land', 'landed'])
```



Load the dataframe

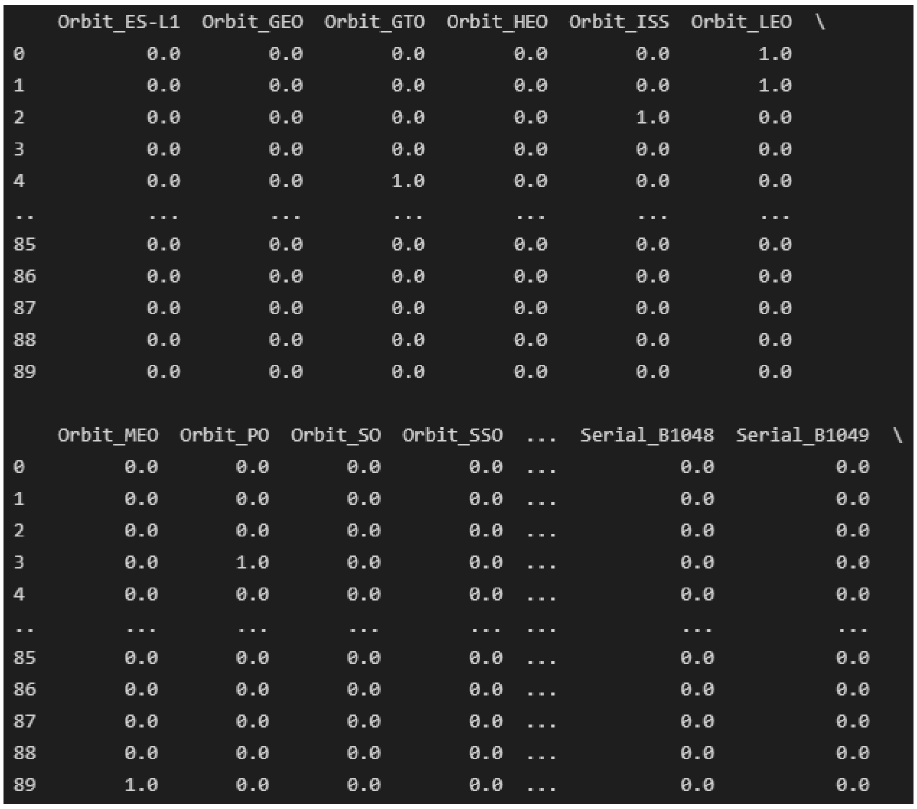
Load the data

```
data = pd.read_csv("dataset_part_2.csv")

# data = pd.read_csv("https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/dataset_part_2.csv")

# If you were unable to complete the previous lab correctly you can uncomment and load this csv

# data = pd.read_csv('https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-DS0701EN-SkillsNetwork/api/dataset_part_2.csv')

data.head()
```



```
   FlightNumber        Date BoosterVersion  PayloadMass Orbit    LaunchSite  \
0             1  2010-06-04       Falcon 9  6123.547647   LEO  CCSFS SLC 40
1             2  2012-05-22       Falcon 9   525.000000   LEO  CCSFS SLC 40
2             3  2013-03-01       Falcon 9   677.000000   ISS  CCSFS SLC 40
3             4  2013-09-29       Falcon 9   500.000000    PO   VAFB SLC 4E
4             5  2013-12-03       Falcon 9  3170.000000   GTO  CCSFS SLC 40

       Outcome  Flights  GridFins  Reused   Legs LandingPad  Block  \
0    None None        1     False   False  False        NaN    1.0
1    None None        1     False   False  False        NaN    1.0
2    None None        1     False   False  False        NaN    1.0
3  False Ocean        1     False   False  False        NaN    1.0
4    None None        1     False   False  False        NaN    1.0

   ReusedCount Serial   Longitude   Latitude  Class
0            0  B0003  -80.577366  28.561857      0
1            0  B0005  -80.577366  28.561857      0
2            0  B0007  -80.577366  28.561857      0
3            0  B1003 -120.610829  34.632093      0
4            0  B1004  -80.577366  28.561857      0
```

```
X = pd.read_csv('dataset_part_3.csv')
# X = pd.read_csv('https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/dataset_part_3.csv')

# If you were unable to complete the previous lab correctly you can uncomment and load this csv

# X = pd.read_csv('https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-DS0701EN-SkillsNetwork/api/dataset_part_3.csv')

X.head(100)
```



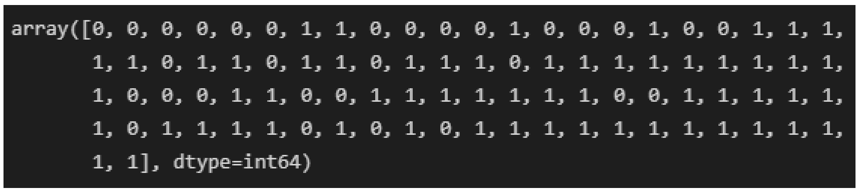
```
    Orbit_ES-L1  Orbit_GEO  Orbit_GTO  Orbit_HEO  Orbit_ISS  Orbit_LEO  \
0           0.0        0.0        0.0        0.0        0.0        1.0
1           0.0        0.0        0.0        0.0        0.0        1.0
2           0.0        0.0        0.0        0.0        1.0        0.0
3           0.0        0.0        0.0        0.0        0.0        0.0
4           0.0        0.0        1.0        0.0        0.0        0.0
..          ...        ...        ...        ...        ...        ...
85          0.0        0.0        0.0        0.0        0.0        0.0
86          0.0        0.0        0.0        0.0        0.0        0.0
87          0.0        0.0        0.0        0.0        0.0        0.0
88          0.0        0.0        0.0        0.0        0.0        0.0
89          0.0        0.0        0.0        0.0        0.0        0.0

    Orbit_MEO  Orbit_PO  Orbit_SO  Orbit_SSO  ...  Serial_B1048  Serial_B1049  \
0         0.0       0.0       0.0        0.0  ...           0.0           0.0
1         0.0       0.0       0.0        0.0  ...           0.0           0.0
2         0.0       0.0       0.0        0.0  ...           0.0           0.0
3         0.0       1.0       0.0        0.0  ...           0.0           0.0
4         0.0       0.0       0.0        0.0  ...           0.0           0.0
..        ...       ...       ...        ...  ...           ...           ...
85        0.0       0.0       0.0        0.0  ...           0.0           0.0
86        0.0       0.0       0.0        0.0  ...           0.0           0.0
87        0.0       0.0       0.0        0.0  ...           0.0           0.0
88        0.0       0.0       0.0        0.0  ...           0.0           0.0
89        1.0       0.0       0.0        0.0  ...           0.0           0.0
```

TASK 1: create a NumPy array from the column Class in data →as Y

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

```
Y = data.Class.to_numpy()
```

```
Y
```



```
array([0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1,
       1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1,
       1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1], dtype=int64)
```

TASK 2: standardize the data in X

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

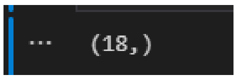
```
transform = preprocessing.StandardScaler()

X = transform.fit(X).transform(X)
```

TASK 3: split the data into training and testing data → train_test_split

We split the data into training and testing data using the function train_test_split. The training data is divided into validation data, a second set used for training data; then the models are trained and hyperparameters are selected using the function GridSearchCV.

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

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

Y_test.shape
```



```
...   (18,)
```

we can see we only have 18 test samples (= 20% samples).

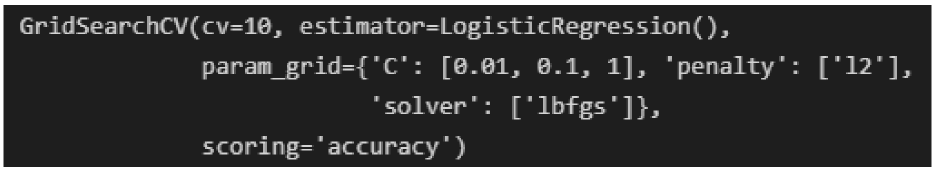
Task 4: create a logistic regression object

Create a logistic regression object, then create a GridSearchCV object logreg_cv with cv = 10. Fit the object to find the best parameters from the dictionary parameters.

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

```
lr=LogisticRegression()

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

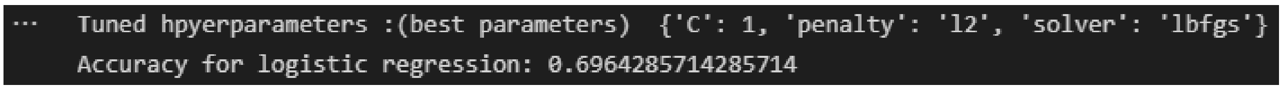
→kết quả:



```
GridSearchCV(cv=10, estimator=LogisticRegression(),
             param_grid={'C': [0.01, 0.1, 1], 'penalty': ['l2'],
                         'solver': ['lbfgs']},
             scoring='accuracy')
```

We output the GridSearchCV object for logistic regression. We display the best parameters using the data attribute best_params_ and the accuracy on the validation data using the data attribute best_score_.

```
print("Tuned hpyerparameters :(best parameters) ", logreg_cv.best_params_)
print("Accuracy for logistic regression:", logreg_cv.best_score_)
```



```
...   Tuned hpyerparameters :(best parameters)  {'C': 1, 'penalty': 'l2', 'solver': 'lbfgs'}
      Accuracy for logistic regression: 0.6964285714285714
```

Task 5: calculate the accuracy on the test data → 0.778

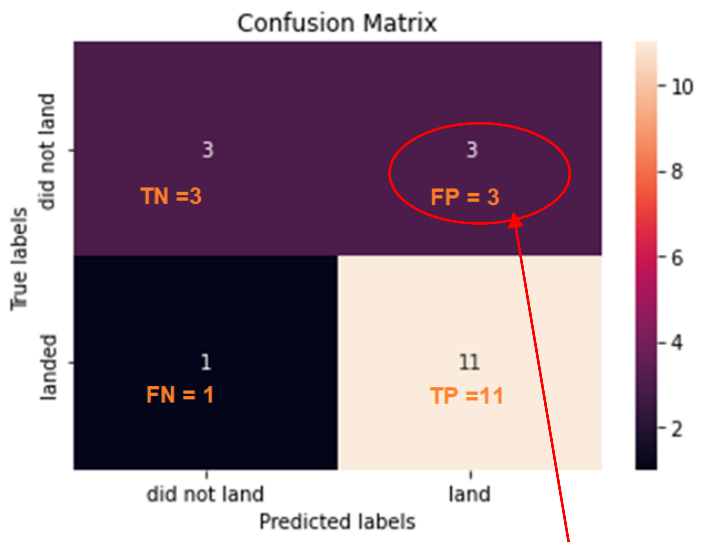
Calculate the accuracy on the test data using the method score:

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

→ 0.7777777777777778

Let's look at the confusion matrix:

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



Accuracy: (TP+TN)/Total = (11+3)/18 = **0.778**
Precision: TP/ Predicted Yes = 11/ (11+3) = 0.786
Misclassification Rate: (FN+FP)/Total = (1+3)/18 = 0.222
True Positive Rate: TP/ Actual Yes = 11/(1+11) = 0.916
False Positive Rate: FP/Actual No = 3/(3+3)= 0.5
True Negative Rate: TN/Actual No = 3/(3+3)= 0.5
Prevalence: Actual Yes/ Total = (1+11)/18 = 0.667

Accuracy # Precision

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

Task 6: create a support vector machine object

Create a support vector machine object then create a GridSearchCV object svm_cv with cv = 10. Fit the object to find the best parameters from the dictionary parameters.

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

```
svm = SVC()
svm_cv = GridSearchCV(svm, parameters, cv=10)
svm_cv.fit(X_train, Y_train)
```

```
GridSearchCV(cv=10, estimator=SVC(),
             param_grid={'C': array([1.00000000e-03, 3.16227766e-02, 1.00000000e+00, 3.16227766e+01,
       1.00000000e+03]),
                         'gamma': array([1.00000000e-03, 3.16227766e-02, 1.00000000e+00, 3.16227766e+01,
       1.00000000e+03]),
                         'kernel': ['sigmoid']})
```

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

```
tuned hpyerparameters :(best parameters)  {'C': 31.622776601683793, 'gamma': 0.03162277660168379, 'kernel': 'sigmoid'}
accuracy : 0.7964285714285715
```

Task 7: calculate the accuracy using the method score → 0.944

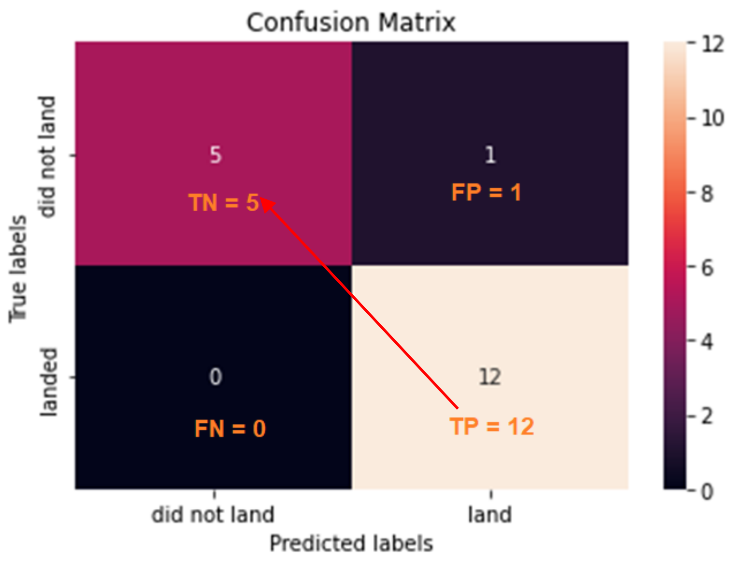
Calculate the accuracy on the test data using the method score:

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

→ 0.9444444444444444

We can plot the confusion matrix

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



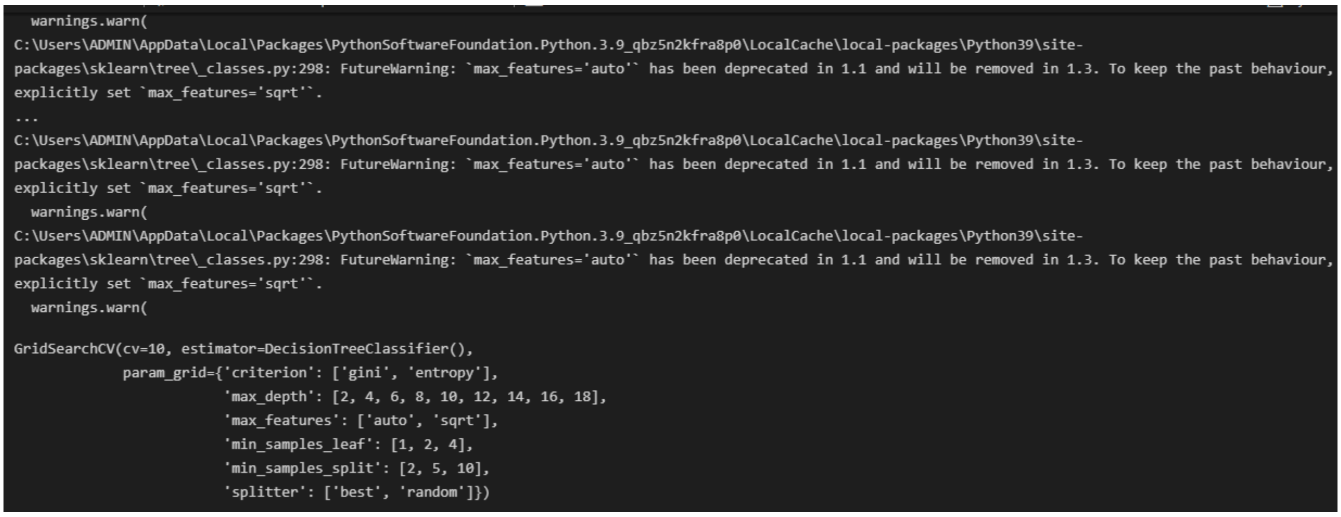
Accuracy: (TP+TN)/Total = (12+5)/18 = **0.944**

Task 8: create a decision tree classifier object

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

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

```
tree = DecisionTreeClassifier()
tree_cv = GridSearchCV(tree, parameters, cv=10)
tree_cv.fit(X_train, Y_train)
```



```
  warnings.warn(
C:\Users\ADMIN\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.9_qbz5n2kfra8p0\LocalCache\local-packages\Python39\site-
packages\sklearn\tree\_classes.py:298: FutureWarning: `max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour,
explicitly set `max_features='sqrt'`.
...
C:\Users\ADMIN\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.9_qbz5n2kfra8p0\LocalCache\local-packages\Python39\site-
packages\sklearn\tree\_classes.py:298: FutureWarning: `max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour,
explicitly set `max_features='sqrt'`.
  warnings.warn(
C:\Users\ADMIN\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.9_qbz5n2kfra8p0\LocalCache\local-packages\Python39\site-
packages\sklearn\tree\_classes.py:298: FutureWarning: `max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour,
explicitly set `max_features='sqrt'`.
  warnings.warn(

GridSearchCV(cv=10, estimator=DecisionTreeClassifier(),
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': [2, 4, 6, 8, 10, 12, 14, 16, 18],
                         'max_features': ['auto', 'sqrt'],
                         'min_samples_leaf': [1, 2, 4],
                         'min_samples_split': [2, 5, 10],
                         'splitter': ['best', 'random']})
```

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

```
tuned hpyerparameters :(best parameters)  {'criterion': 'entropy', 'max_depth': 16, 'max_features': 'sqrt', 'min_samples_leaf': 1, 'min_samples_split': 10,
'splitter': 'best'}
accuracy : 0.7964285714285715
```

Task 9: calculate the accuracy of tree_cv → 0.667

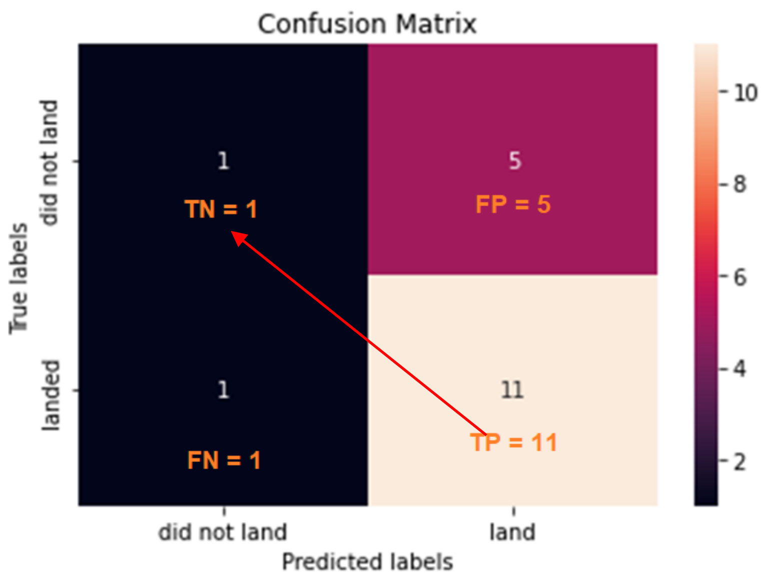
Calculate the accuracy of tree_cv on the test data using the method score:

```
tree_score = tree_cv.score(X_test, Y_test)
tree_score
```

→ 0.6666666666666666

We can plot the confusion matrix.

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



Accuracy:
(TP+TN)/Total = (11+1)/18 = **0.667**

Task 10: create a k-nearest neighbors object

Create a k nearest neighbors object then create a GridSearchCV object knn_cv with cv = 10. Fit the object to find the best parameters from the dictionary parameters.

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

```
knn = KNeighborsClassifier()
knn_cv = GridSearchCV(knn, parameters, scoring='accuracy', cv=10)
knn_cv.fit(X_train, Y_train)
```

```
GridSearchCV(cv=10, estimator=KNeighborsClassifier(),
             param_grid={'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],
                         'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
                         'p': [1, 2]},
             scoring='accuracy')
```

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

```
tuned hpyerparameters :(best parameters)  {'algorithm': 'auto', 'n_neighbors': 10, 'p': 1}
accuracy : 0.7392857142857142
```

Task 11: calculate the accuracy of knn_cv → 0.722

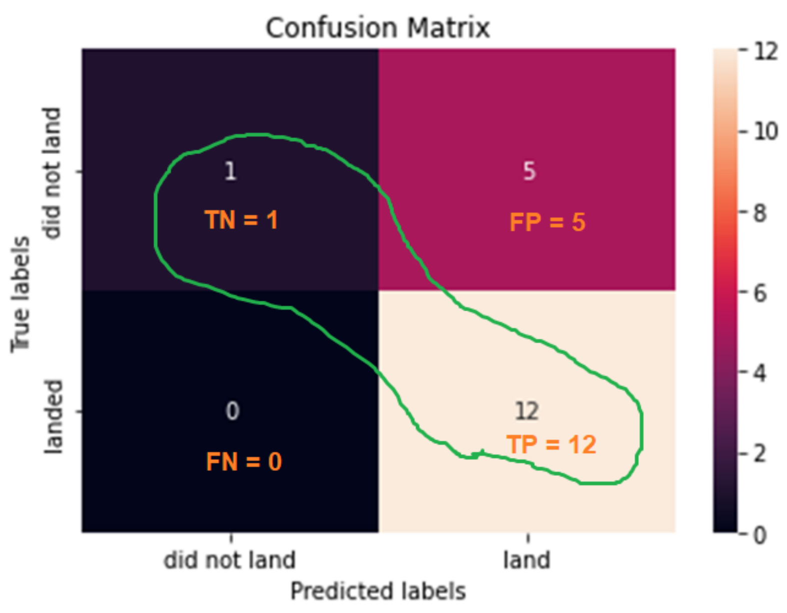
Calculate the accuracy of knn_cv on the test data using the method score:

```
knn_score = knn_cv.score(X_test, Y_test)
knn_score
```

→ 0.7222222222222222

We can plot the confusion matrix.

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



Accuracy:
(TP+TN)/Total = (12+1)/18 = **0.722**

Task 12: Find the method performs best

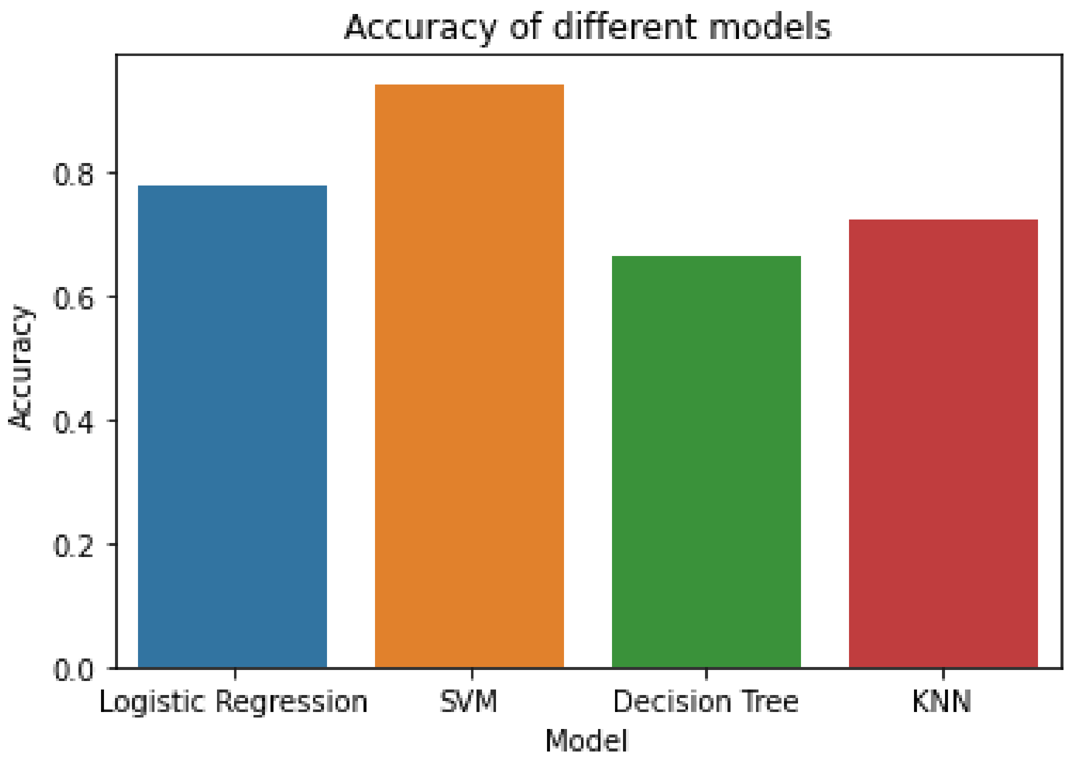
```
scores = {'Logistic Regression': logreg_score, 'SVM': svm_score, 'Decision Tree':
tree_score, 'KNN': knn_score}

#scores = {'Logistic Regression': logreg_cv.best_score_, 'SVM': svm_cv.best_score_,
'Decision Tree': tree_cv.best_score_, 'KNN': knn_cv.best_score_}
print("The best model is: ", max(scores, key=scores.get), "with the accuracy of ",
round(max(scores.values())*100,2), "%")
```

→ The best model is: SVM with the accuracy of 94.44 %

```
algo_df = pd.DataFrame.from_dict(scores, orient='index', columns=['Accuracy'])
algo_df.head()
```

```
                     Accuracy
Logistic Regression  0.777778
SVM                  0.944444
Decision Tree        0.666667
KNN                  0.722222
```

```
sns.barplot(x=list(scores.keys()), y=list(scores.values()))
plt.title("Accuracy of different models")
plt.xlabel("Model")
plt.ylabel("Accuracy")
```



tham khảo vẽ biểu đồ barchart: thêm label

https://phamdinhkhanh.github.io/deepai-book/ch_appendix/appendix_matplotlib.html#barchart