```
import json
import requests
import pandas as pd


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


```
        'ships': ['5ea6ed2d080df4000697c901'],
        'static_fire_date_unix': 1291420800,
        'static_fire_date_utc': '2010-12-04T00:00:00.000Z',
        'success': True,
        'tbd': False,
        'upcoming': False,
        'window': 0},
       {'auto_update': True,
        'capsules': ['5e9e2c5bf3591882af3b2665'],
        'cores': [{'core': '5e9e289ef35918f39c3b262a',
          'flight': 1,
          'gridfins': False,
          'landing_attempt': False,
          'landing_success': None,
          'landing_type': None,
          'landpad': None,
          'legs': False,
          'reused': False}],
        'crew': [],
        'date_local': '2012-05-22T03:44:00-04:00',
        'date_precision': 'hour',
        'date_unix': 1335944640,
        'date_utc': '2012-05-22T07:44:00.000Z',
        'details': 'Launch was scrubbed on first attempt, second launch attempt was
      successful',
        'failures': [],
        'fairings': None,
        'flight_number': 8,
        'id': '5eb87cdfffd86e000604b331',
        'launch_library_id': None,
        'launchpad': '5e9e4501f509094ba4566f84',
        'links': {'article': 'https://en.wikipedia.org/wiki/Dragon_C2%2B',
         'flickr': {'original': [], 'small': []},
         'patch': {'large': 'https://images2.imgbox.com/2b/8e/MYyHbnd2_o.png',
          'small': 'https://images2.imgbox.com/fc/7a/r9ITwL12_o.png'},
         'presskit': 'https://www.nasa.gov/pdf/649910main_cots2_presskit_051412.pdf',
         'reddit': {'campaign': None,
          'launch': None,
          'media': None,
          'recovery': None},
         'webcast': 'https://www.youtube.com/watch?v=tpQzDbAY7yI',
         'wikipedia': 'https://en.wikipedia.org/wiki/Dragon_C2%2B',
         'youtube_id': 'tpQzDbAY7yI'},
        'name': 'COTS 2',
        'net': False,
        'payloads': ['5eb0e4bab6c3bb0006eeb1ea'],
        'rocket': '5e9d0d95eda69973a809d1ec',
        'ships': ['5ea6ed2d080df4000697c901'],
        'static_fire_date_unix': 1335744000,
        'static_fire_date_utc': '2012-04-30T00:00:00.000Z',
```

```
 'success': True,
 'tbd': False,
 'upcoming': False,
 'window': 0},
{'auto_update': True,
 'capsules': ['5e9e2c5bf3591835983b2666'],
 'cores': [{'core': '5e9e289ff3591821a73b262b',
   'flight': 1
```

```
#converting json object to structured table data
data=pd.json_normalize(response.json())
```

```
data
```

	static_fire_date_utc	static_fire_date_unix	net	window	r
0	2006-03-				

```
data.head()
```

	static_fire_date_utc	static_fire_date_unix	net	window	rock
0	2006-03-17T00:00:00.000Z	1.142554e+09	False	0.0	5e9d0d95eda69955f709d1
1	None	NaN	False	0.0	5e9d0d95eda69955f709d1
2	None	NaN	False	0.0	5e9d0d95eda69955f709d1
3	2008-09-20T00:00:00.000Z	1.221869e+09	False	0.0	5e9d0d95eda69955f709d1
4	None	NaN	False	0.0	5e9d0d95eda69955f709d1

5 rows × 43 columns

```
data.describe()
```

	static_fire_date_unix	window	flight_number	date_unix	fairings
count	1.210000e+02	117.000000	179.000000	1.790000e+02	0.0
mean	1.520206e+09	2568.974359	90.000000	1.551486e+09	NaN
std	9.080036e+07	4389.948430	51.816986	1.031655e+08	NaN

```
# Requests allows us to make HTTP requests which we will use to get data from an API
import requests
# Pandas is a software library written for the Python programming language for data manipu
import pandas as pd
# NumPy is a library for the Python programming language, adding support for large, multi-
import numpy as np
# Datetime is a library that allows us to represent dates
import datetime

# Setting this option will print all collumns of a dataframe
pd.set_option('display.max_columns', None)
# Setting this option will print all of the data in a feature
pd.set_option('display.max_colwidth', None)
```

```
# Takes the dataset and uses the rocket column to call the API and append the data to the
def getBoosterVersion(data):
    for x in data['rocket']:
        response = requests.get("https://api.spacexdata.com/v4/rockets/"+str(x)).json()
        BoosterVersion.append(response['name'])
```

```
# Takes the dataset and uses the launchpad column to call the API and append the data to t
def getLaunchSite(data):
    for x in data['launchpad']:
        response = requests.get("https://api.spacexdata.com/v4/launchpads/"+str(x)).json()
        Longitude.append(response['longitude'])
        Latitude.append(response['latitude'])
        LaunchSite.append(response['name'])
```

```
# Takes the dataset and uses the payloads column to call the API and append the data to th
def getPayloadData(data):
    for load in data['payloads']:
        response = requests.get("https://api.spacexdata.com/v4/payloads/"+load).json()
        PayloadMass.append(response['mass_kg'])
        Orbit.append(response['orbit'])
```

```
# Takes the dataset and uses the cores column to call the API and append the data to the l
def getCoreData(data):
    for core in data['cores']:
            if core['core'] != None:
                response = requests.get("https://api.spacexdata.com/v4/cores/"+core['core'
                Block.append(response['block'])
                ReusedCount.append(response['reuse_count'])
```

```
                Serial.append(response['serial'])
            else:
                Block.append(None)
                ReusedCount.append(None)
                Serial.append(None)
            Outcome.append(str(core['landing_success'])+' '+str(core['landing_type']))
            Flights.append(core['flight'])
            GridFins.append(core['gridfins'])
            Reused.append(core['reused'])
            Legs.append(core['legs'])
            LandingPad.append(core['landpad'])
```

```
spacex_url="https://api.spacexdata.com/v4/launches/past"
```

```
response = requests.get(spacex_url)
```

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

```
b'[{"fairings":{"reused":false,"recovery_attempt":false,"recovered":false,"ships":[]}
```

```
static_json_url='https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS
```

```
response.status_code
```

```
200
```

```
# Use json_normalize meethod to convert the json result into a dataframe
data = pd.json_normalize(response.json())
```

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

	static_fire_date_utc	static_fire_date_unix	net	window	rock
0	2006-03-17T00:00:00.000Z	1.142554e+09	False	0.0	5e9d0d95eda69955f709d1
1	None	NaN	False	0.0	5e9d0d95eda69955f709d1
2	None	NaN	False	0.0	5e9d0d95eda69955f709d1
3	2008-09-20T00:00:00.000Z	1.221869e+09	False	0.0	5e9d0d95eda69955f709d1

```
# Lets take a subset of our dataframe keeping only the features we want and the flight num
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 ro
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 th
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
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)]


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


BoosterVersion
```

```
[]
```

```
# Call getBoosterVersion
getBoosterVersion(data)


BoosterVersion[0:5]
```

```
['Falcon 1', 'Falcon 1', 'Falcon 1', 'Falcon 1', 'Falcon 9']
```

```
# Call getLaunchSite
getLaunchSite(data)


# Call getPayloadData
getPayloadData(data)


# Call getCoreData
getCoreData(data)
```

```
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 data from launch_dict
data = pd.DataFrame(launch_dict)


# Show the head of the dataframe
data
```

FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights
1	2006-03-24	Falcon 1	20.0	LEO	Kwajalein Atoll	None None	1
2	2007-03-21	Falcon 1	NaN	LEO	Kwajalein Atoll	None None	1

```
data.head()
```

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flig
0	1	2006-03-24	Falcon 1	20.0	LEO	Kwajalein Atoll	None None	
1	2	2007-03-21	Falcon 1	NaN	LEO	Kwajalein Atoll	None None	
2	4	2008-09-28	Falcon 1	165.0	LEO	Kwajalein Atoll	None None	
3	5	2009-07-13	Falcon 1	200.0	LEO	Kwajalein Atoll	None None	
4	6	2010-06-04	Falcon 9	NaN	LEO	CCSFS SLC 40	None None	

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

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Fli
4	6	2010-06-04	Falcon 9	NaN	LEO	CCSFS SLC 40	None None	
5	8	2012-05-22	Falcon 9	525.0	LEO	CCSFS SLC 40	None None	
6	10	2013-03-01	Falcon 9	677.0	ISS	CCSFS SLC 40	None None	
7	11	2013-09-29	Falcon 9	500.0	PO	VAFB SLC 4E	False Ocean	

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

```
/usr/local/lib/python3.7/dist-packages/pandas/core/indexing.py:1773: SettingWithCopyW
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/us
  self._setitem_single_column(ilocs[0], value, pi)
(90, 17)
```

10-18 39A ASDS

Data Wrangling

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

```
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
```

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

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:4: SettingWithCopyWarnir
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/us
  after removing the cwd from sys.path.
```

```
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
```

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

week-2

```
!pip install sqlalchemy==1.3.9
!pip install ibm_db_sa
!pip install ipython-sql
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/p
Collecting sqlalchemy==1.3.9
  Downloading SQLAlchemy-1.3.9.tar.gz (6.0 MB)
     |████████████████████████████████| 6.0 MB 6.4 MB/s
Building wheels for collected packages: sqlalchemy
  Building wheel for sqlalchemy (setup.py) ... done
  Created wheel for sqlalchemy: filename=SQLAlchemy-1.3.9-cp37-cp37m-linux_x86_64.whl
  Stored in directory: /root/.cache/pip/wheels/03/71/13/010faf12246f72dc76b4150e6e59
Successfully built sqlalchemy
Installing collected packages: sqlalchemy
  Attempting uninstall: sqlalchemy
    Found existing installation: SQLAlchemy 1.4.40
    Uninstalling SQLAlchemy-1.4.40:
      Successfully uninstalled SQLAlchemy-1.4.40
```

```
Successfully installed sqlalchemy-1.3.9
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/p
Collecting ibm_db_sa
  Downloading ibm_db_sa-0.3.8-py3-none-any.whl (30 kB)
Requirement already satisfied: sqlalchemy>=0.7.3 in /usr/local/lib/python3.7/dist-pac
Collecting ibm-db>=2.0.0
  Downloading ibm_db-3.1.3.tar.gz (1.4 MB)
     |████████████████████████████████| 1.4 MB 11.6 MB/s
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Installing backend dependencies ... done
    Preparing wheel metadata ... done
Building wheels for collected packages: ibm-db
  Building wheel for ibm-db (PEP 517) ... done
  Created wheel for ibm-db: filename=ibm_db-3.1.3-cp37-cp37m-linux_x86_64.whl size=41
  Stored in directory: /root/.cache/pip/wheels/a7/fe/6f/52ae8e5a30a0626cec5f28f908e4c
Successfully built ibm-db
Installing collected packages: ibm-db, ibm-db-sa
Successfully installed ibm-db-3.1.3 ibm-db-sa-0.3.8
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/p
Requirement already satisfied: ipython-sql in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: sqlalchemy>=0.6.7 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages (from ip
Requirement already satisfied: ipython>=1.0 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: sqlparse in /usr/local/lib/python3.7/dist-packages (fr
Requirement already satisfied: prettytable in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: ipython-genutils>=0.1.0 in /usr/local/lib/python3.7/di
Requirement already satisfied: prompt-toolkit<2.0.0,>=1.0.4 in /usr/local/lib/python3
Requirement already satisfied: simplegeneric>0.8 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: pygments in /usr/local/lib/python3.7/dist-packages (fr
Requirement already satisfied: traitlets>=4.2 in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: pexpect in /usr/local/lib/python3.7/dist-packages (fro
Requirement already satisfied: setuptools>=18.5 in /usr/local/lib/python3.7/dist-pack
Requirement already satisfied: pickleshare in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: decorator in /usr/local/lib/python3.7/dist-packages (f
Requirement already satisfied: wcwidth in /usr/local/lib/python3.7/dist-packages (fro
Requirement already satisfied: ptyprocess>=0.5 in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: importlib-metadata in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.7/dist-packages (f
Requirement already satisfied: typing-extensions>=3.6.4 in /usr/local/lib/python3.7/d
```

```
%load_ext sql
```

```
%sql ibm_db_sa://sdk38546:cwn1%40l380qx5qb3k@dashdb-txn-sbox-yp-lon02-07.services.eu-gb.bl
```

```
Connection info needed in SQLAlchemy format, example:
               postgresql://username:password@hostname/dbname
               or an existing connection: dict_keys([])
Can't load plugin: sqlalchemy.dialects:ibm_db_sa
Connection info needed in SQLAlchemy format, example:
               postgresql://username:password@hostname/dbname
               or an existing connection: dict_keys([])
```

```
%sql select distinct(LAUNCH_SITE) from SPACEXTBL
```

```
     Environment variable $DATABASE_URL not set, and no connect string given.
     Connection info needed in SQLAlchemy format, example:
                    postgresql://username:password@hostname/dbname
                    or an existing connection: dict_keys([])
```

```
%sql select * from SPACEXTBL where LAUNCH_SITE like 'CCA%' limit 5
```

```
     Environment variable $DATABASE_URL not set, and no connect string given.
     Connection info needed in SQLAlchemy format, example:
                    postgresql://username:password@hostname/dbname
                    or an existing connection: dict_keys([])
```

```
%sql select sum(PAYLOAD_MASS__KG_) from SPACEXTBL where CUSTOMER = 'NASA (CRS)'
```

```
     Environment variable $DATABASE_URL not set, and no connect string given.
     Connection info needed in SQLAlchemy format, example:
                    postgresql://username:password@hostname/dbname
                    or an existing connection: dict_keys([])
```

```
%sql select avg(PAYLOAD_MASS__KG_) from SPACEXTBL where BOOSTER_VERSION = 'F9 v1.1'
```

```
     Environment variable $DATABASE_URL not set, and no connect string given.
     Connection info needed in SQLAlchemy format, example:
                    postgresql://username:password@hostname/dbname
                    or an existing connection: dict_keys([])
```

```
%sql select min(DATE) from SPACEXTBL where Landing__Outcome = 'Success (ground pad)'
```

```
     Environment variable $DATABASE_URL not set, and no connect string given.
     Connection info needed in SQLAlchemy format, example:
                    postgresql://username:password@hostname/dbname
                    or an existing connection: dict_keys([])
```

```
%sql select BOOSTER_VERSION from SPACEXTBL where Landing__Outcome = 'Success (drone ship)'
```

```
     Environment variable $DATABASE_URL not set, and no connect string given.
     Connection info needed in SQLAlchemy format, example:
                    postgresql://username:password@hostname/dbname
                    or an existing connection: dict_keys([])
```

```
%sql select count(MISSION_OUTCOME) from SPACEXTBL where MISSION_OUTCOME = 'Success' or MIS
```

```
     Environment variable $DATABASE_URL not set, and no connect string given.
     Connection info needed in SQLAlchemy format, example:
                    postgresql://username:password@hostname/dbname
                    or an existing connection: dict_keys([])
```

```
%sql select BOOSTER_VERSION from SPACEXTBL where PAYLOAD_MASS__KG_ = (select max(PAYLOAD_M
```

```
     Environment variable $DATABASE_URL not set, and no connect string given.
     Connection info needed in SQLAlchemy format, example:
```

```
                    postgresql://username:password@hostname/dbname
                    or an existing connection: dict_keys([])
```

```
%sql SELECT EXTRACT(MONTH, select min(DATE) from SPACEXTBL where Landing__Outcome = 'Succe
```

```
     Environment variable $DATABASE_URL not set, and no connect string given.
     Connection info needed in SQLAlchemy format, example:
                    postgresql://username:password@hostname/dbname
                    or an existing connection: dict_keys([])
```

```
%sql select * from SPACEXTBL where Landing__Outcome like 'Success%' and (DATE between '201
```

```
     Environment variable $DATABASE_URL not set, and no connect string given.
     Connection info needed in SQLAlchemy format, example:
                    postgresql://username:password@hostname/dbname
                    or an existing connection: dict_keys([])
```

week-3(interactive visual analytics)

```
# andas is a software library written for the Python programming language for data manipul
import pandas as pd
#NumPy is a library for the Python programming language, adding support for large, multi-d
import numpy as np
# Matplotlib is a plotting library for python and pyplot gives us a MatLab like plotting f
import matplotlib.pyplot as plt
#Seaborn is a Python data visualization library based on matplotlib. It provides a high-le
import seaborn as sns
```

```
df=pd.read_csv("https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0
```

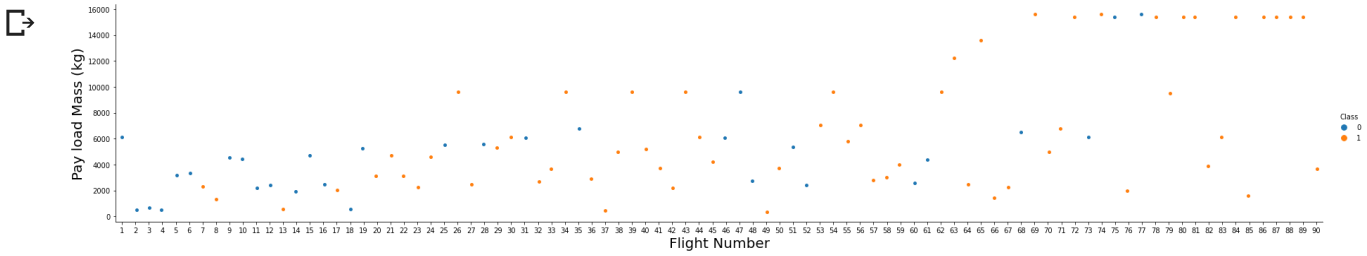
```
# If you were unable to complete the previous lab correctly you can uncomment and load thi
```

```
# df = pd.read_csv('https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM
```

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

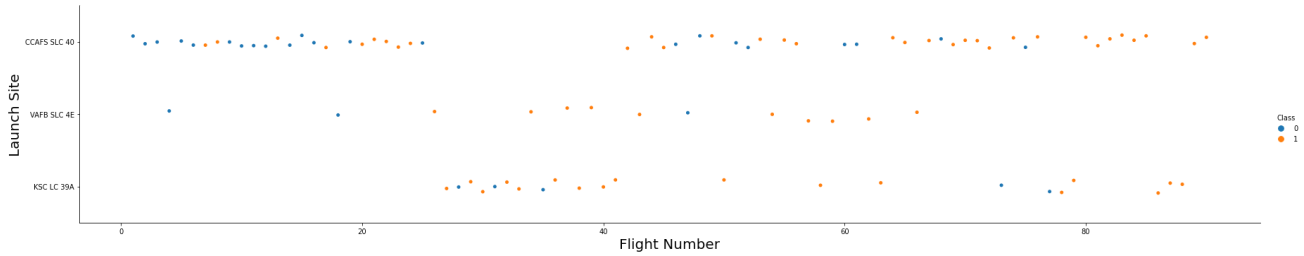
	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flig
		2010-				CCAFS	None	
		2012-				CCAFS	None	

```
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()
```



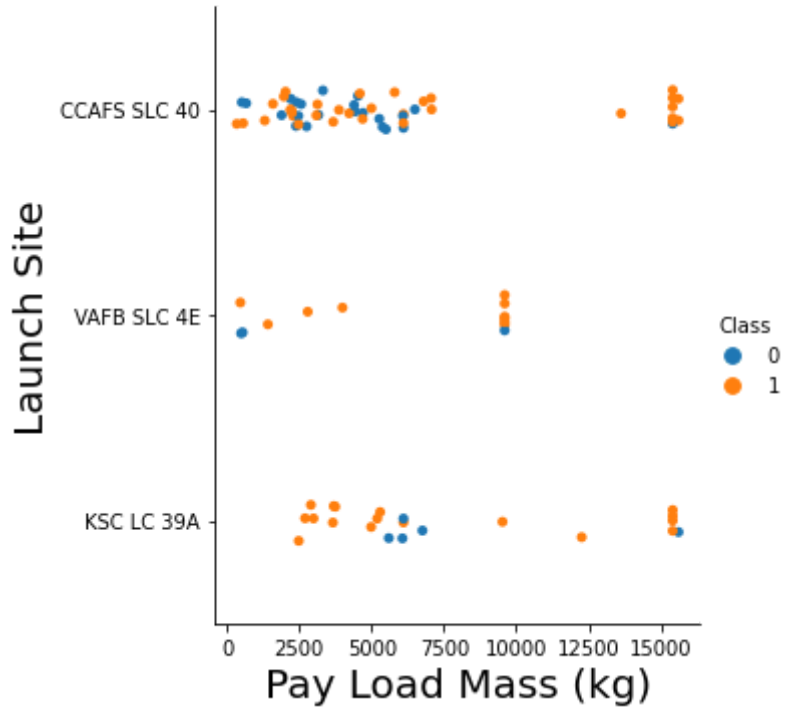
TASK 1: Visualize the relationship between Flight Number and Launch Site

```
# Plot a scatter point chart with x axis to be Flight Number and y axis to be the launch s
sns.catplot(y="LaunchSite", x="FlightNumber", hue="Class", data=df, aspect = 5)
plt.xlabel("Flight Number",fontsize=20)
plt.ylabel("Launch Site",fontsize=20)
plt.show()
```

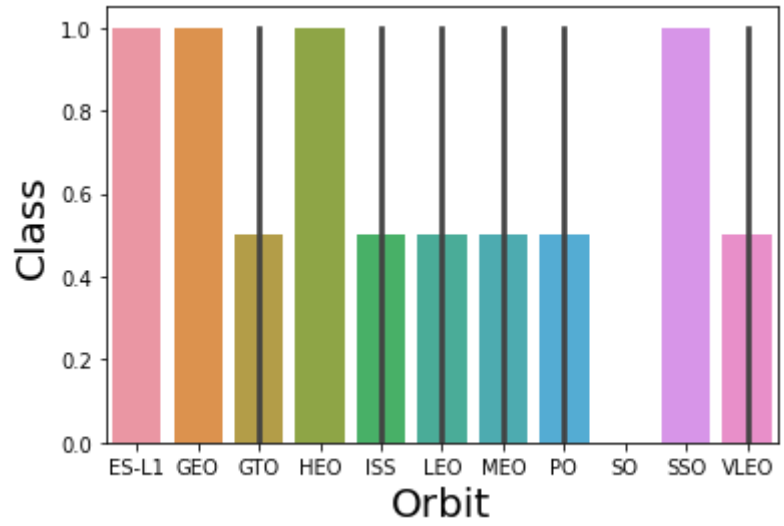



```
# Plot a scatter point chart with x axis to be Pay Load Mass (kg) and y axis to be the lau
sns.catplot(y="LaunchSite", x="PayloadMass", hue="Class", data=df)
```

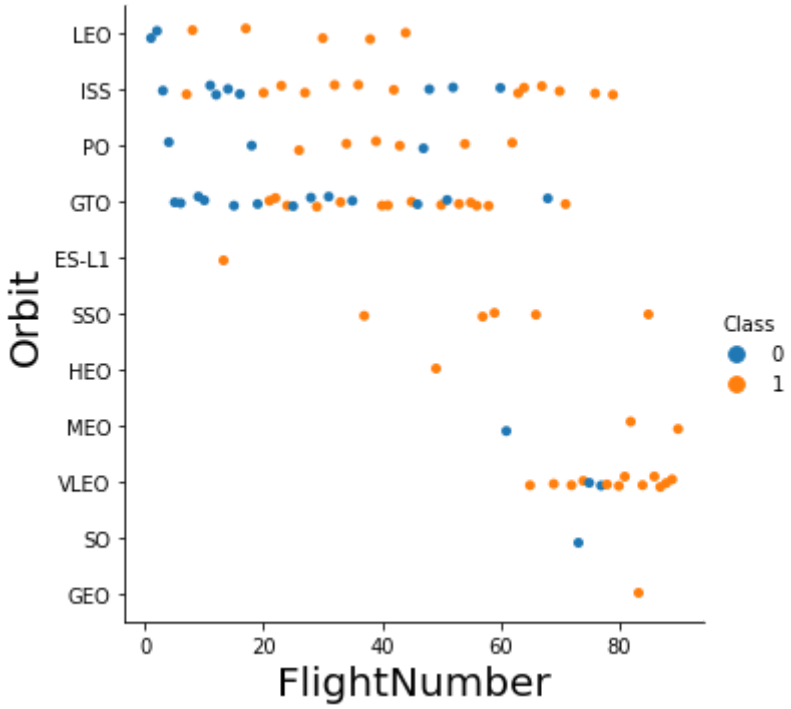
```
plt.xlabel("Pay Load Mass (kg)",fontsize=20)
plt.ylabel("Launch Site",fontsize=20)
plt.show()
```



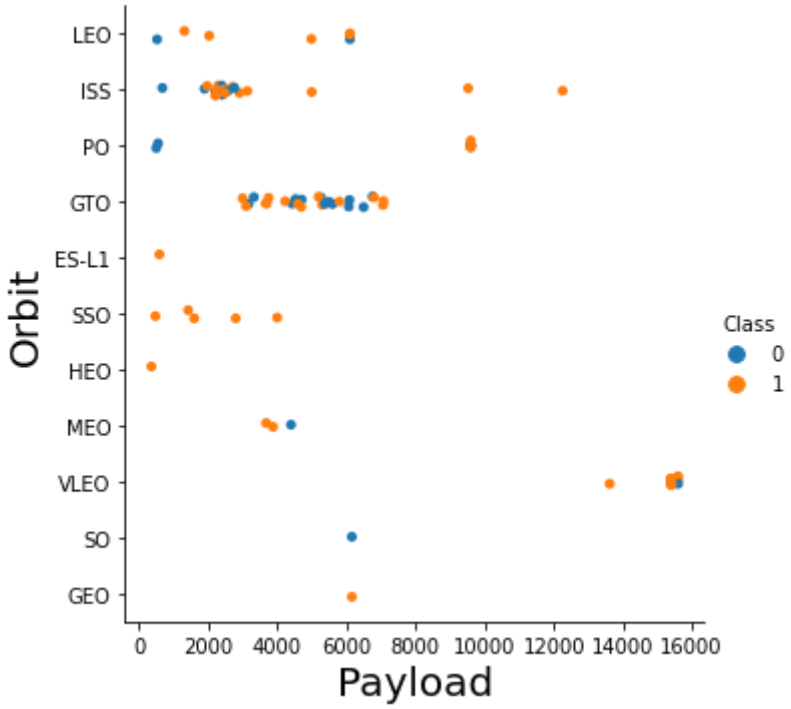

```
# HINT use groupby method on Orbit column and get the mean of Class column
t = df.groupby(['Orbit', 'Class'])['Class'].agg(['mean']).reset_index()
sns.barplot(y="Class", x="Orbit", data=t)

plt.xlabel("Orbit",fontsize=20)
plt.ylabel("Class",fontsize=20)
plt.show()
```




```
# Plot a scatter point chart with x axis to be FlightNumber and y axis to be the Orbit, an
sns.catplot(y="Orbit", x="FlightNumber", hue="Class", data=df)
plt.xlabel("FlightNumber",fontsize=20)
plt.ylabel("Orbit",fontsize=20)
plt.show()
```




```
# Plot a scatter point chart with x axis to be Payload and y axis to be the Orbit, and hue
sns.catplot(y="Orbit", x="PayloadMass", hue="Class", data=df)
plt.xlabel("Payload",fontsize=20)
plt.ylabel("Orbit",fontsize=20)
plt.show()
```

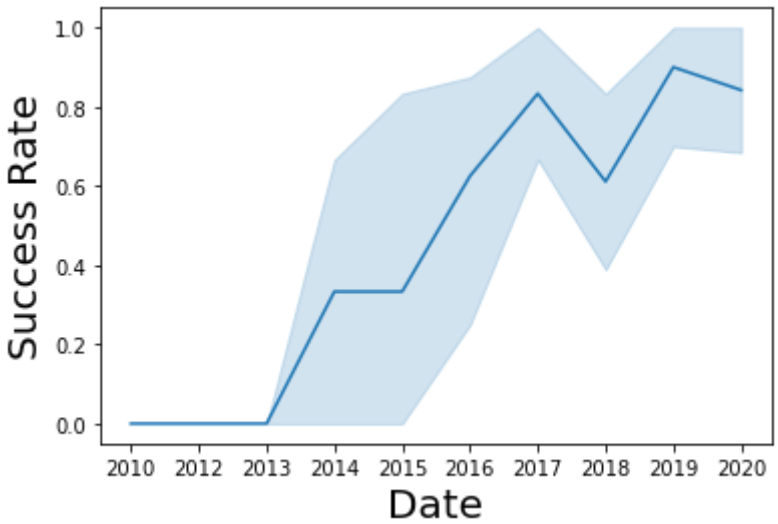



```
# A function to Extract years from the date
def Extract_year():
    for i in df["Date"]:
        year.append(i.split("-")[0])
    return year


year=[]
df1 = df.copy()
```

```
year = Extract_year()
df1["Date"] = year
df1.head()
```

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Fligł
0	1	2010	Falcon 9	6104.959412	LEO	CCAFS SLC 40	None None	
1	2	2012	Falcon 9	525.000000	LEO	CCAFS SLC 40	None None	
2	3	2013	Falcon 9	677.000000	ISS	CCAFS SLC 40	None None	
3	4	2013	Falcon 9	500.000000	PO	VAFB SLC 4E	False Ocean	
4	5	2013	Falcon 9	3170.000000	GTO	CCAFS SLC 40	None None	

```
# Plot a line chart with x axis to be the extracted year and y axis to be the success rate
sns.lineplot(data=df1, x="Date", y="Class")
plt.xlabel("Date",fontsize=20)
plt.ylabel("Success Rate",fontsize=20)
plt.show()
```



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.

```
features = df[['FlightNumber', 'PayloadMass', 'Orbit', 'LaunchSite', 'Flights', 'GridFins'
```

```
features.head()
```

	FlightNumber	PayloadMass	Orbit	LaunchSite	Flights	GridFins	Reused	Legs	La
0	1	6104.959412	LEO	CCAFS SLC 40	1	False	False	False	
1	2	525.000000	LEO	CCAFS SLC 40	1	False	False	False	
2	3	677.000000	ISS	CCAFS SLC 40	1	False	False	False	

TASK 7: Create dummy variables to categorical columns

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

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

	FlightNumber	PayloadMass	Flights	GridFins	Reused	Legs	Block	ReusedCount	0
0	1	6104.959412	1	False	False	False	1.0	0	
1	2	525.000000	1	False	False	False	1.0	0	
2	3	677.000000	1	False	False	False	1.0	0	
3	4	500.000000	1	False	False	False	1.0	0	
4	5	3170.000000	1	False	False	False	1.0	0	

```
#TASK 8: Cast all numeric columns to float64
#Now that our features_one_hot dataframe only contains numbers cast the entire dataframe t

# HINT: use astype function
features_one_hot.astype(float)
```

	FlightNumber	PayloadMass	Flights	GridFins	Reused	Legs	Block	ReusedCount
0	1.0	6104.959412	1.0	0.0	0.0	0.0	1.0	0.0
1	2.0	525.000000	1.0	0.0	0.0	0.0	1.0	0.0
2	3.0	677.000000	1.0	0.0	0.0	0.0	1.0	0.0
3	4.0	500.000000	1.0	0.0	0.0	0.0	1.0	0.0
4	5.0	3170.000000	1.0	0.0	0.0	0.0	1.0	0.0
...	...	...	...	...	...	...	...	...
85	86.0	15400.000000	2.0	1.0	1.0	1.0	5.0	2.0
86	87.0	15400.000000	3.0	1.0	1.0	1.0	5.0	2.0
87	88.0	15400.000000	6.0	1.0	1.0	1.0	5.0	5.0
88	89.0	15400.000000	3.0	1.0	1.0	1.0	5.0	2.0
89	90.0	3681.000000	1.0	1.0	0.0	1.0	5.0	0.0

90 rows × 80 columns

```
features_one_hot.to_csv('dataset_part3.csv', index=False)
```

week-4

predictive analysis

```
# Pandas is a software library written for the Python programming language for data manipu
import pandas as pd
# NumPy is a library for the Python programming language, adding support for large, multi-
import numpy as np
# Matplotlib is a plotting library for python and pyplot gives us a MatLab like plotting f
import matplotlib.pyplot as plt
#Seaborn is a Python data visualization library based on matplotlib. It provides a high-le
import seaborn as sns
# Preprocessing allows us to standarsize 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
```

```
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 l
```

```
data = pd.read_csv("https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM

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

# data = pd.read_csv('https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/I

data.head()
```

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flig
0	1	2010-06-04	Falcon 9	6104.959412	LEO	CCAFS SLC 40	None None	
1	2	2012-05-22	Falcon 9	525.000000	LEO	CCAFS SLC 40	None None	
2	3	2013-03-01	Falcon 9	677.000000	ISS	CCAFS SLC 40	None None	
3	4	2013-09-29	Falcon 9	500.000000	PO	VAFB SLC 4E	False Ocean	
4	5	2013-12-03	Falcon 9	3170.000000	GTO	CCAFS SLC 40	None None	

```
X = pd.read_csv('https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS

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

# X = pd.read_csv('https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMD

X.head(100)
```

	FlightNumber	PayloadMass	Flights	Block	ReusedCount	Orbit_ES-L1	Orbit_GEO	Or
0	1.0	6104.959412	1.0	1.0	0.0	0.0	0.0	
1	2.0	525.000000	1.0	1.0	0.0	0.0	0.0	
2	3.0	677.000000	1.0	1.0	0.0	0.0	0.0	
3	4.0	500.000000	1.0	1.0	0.0	0.0	0.0	
4	5.0	3170.000000	1.0	1.0	0.0	0.0	0.0	
...	...	...	...	...	...	...	...	
85	86.0	15400.000000	2.0	5.0	2.0	0.0	0.0	
86	87.0	15400.000000	3.0	5.0	2.0	0.0	0.0	
87	88.0	15400.000000	6.0	5.0	5.0	0.0	0.0	
88	89.0	15400.000000	3.0	5.0	2.0	0.0	0.0	

```
#TASK 1
#Create a NumPy array from the column Class in data, by applying the method to_numpy() the

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])
```

```
# students get this
transform = preprocessing.StandardScaler()
```

```
X
```

	FlightNumber	PayloadMass	Flights	Block	ReusedCount	Orbit_ES-L1	Orbit_GEO	Or
0	1.0	6104.959412	1.0	1.0	0.0	0.0	0.0	
1	2.0	525.000000	1.0	1.0	0.0	0.0	0.0	
2	3.0	677.000000	1.0	1.0	0.0	0.0	0.0	
3	4.0	500.000000	1.0	1.0	0.0	0.0	0.0	

```
X = transform.fit_transform(X)
X
```

```
array([[-1.71291154e+00, -1.94814463e-16, -6.53912840e-01, ...,
        -8.35531692e-01,  1.93309133e+00, -1.93309133e+00],
       [-1.67441914e+00, -1.19523159e+00, -6.53912840e-01, ...,
        -8.35531692e-01,  1.93309133e+00, -1.93309133e+00],
       [-1.63592675e+00, -1.16267307e+00, -6.53912840e-01, ...,
        -8.35531692e-01,  1.93309133e+00, -1.93309133e+00],
       ...,
       [ 1.63592675e+00,  1.99100483e+00,  3.49060516e+00, ...,
         1.19684269e+00, -5.17306132e-01,  5.17306132e-01],
       [ 1.67441914e+00,  1.99100483e+00,  1.00389436e+00, ...,
         1.19684269e+00, -5.17306132e-01,  5.17306132e-01],
       [ 1.71291154e+00, -5.19213966e-01, -6.53912840e-01, ...,
        -8.35531692e-01, -5.17306132e-01,  5.17306132e-01]])
```

```
#Task-3
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=2)


Y_test.shape
```

```
(18,)
```

```
#task-4
parameters ={'C':[0.01,0.1,1],
             'penalty':['l2'],
             'solver':['lbfgs']}


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

# Instantiate the GridSearchCV object: logreg_cv
logreg_cv = GridSearchCV(lr, parameters, cv=10)

# Fit it to the data
logreg_cv.fit(X_train, Y_train)
```

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

```
print("tuned hpyerparameters :(best parameters) ",logreg_cv.best_params_)
```

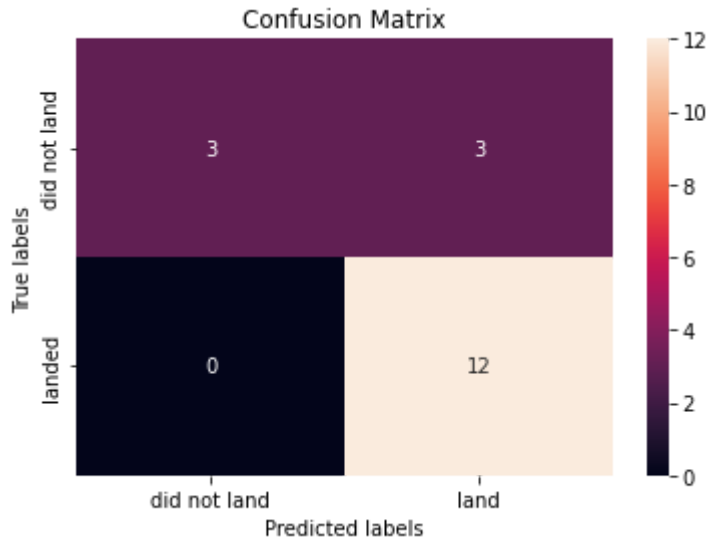
```
print("accuracy :",logreg_cv.best_score_)
```

```
tuned hpyerparameters :(best parameters)  {'C': 0.01, 'penalty': 'l2', 'solver': 'lbf
accuracy : 0.8464285714285713
```

```
#task-5
logreg_cv.score(X_test, Y_test)
```

```
0.8333333333333334
```

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




```
#task-6
parameters = {'kernel':('linear', 'rbf','poly','rbf', 'sigmoid'),
              'C': np.logspace(-3, 3, 5),
              'gamma':np.logspace(-3, 3, 5)}
svm = SVC()
```

```
svm_cv = GridSearchCV(svm, parameters, cv=10)

# Fit it to the data
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': ('linear', 'rbf', 'poly', 'rbf', 'sigmoid')})
```

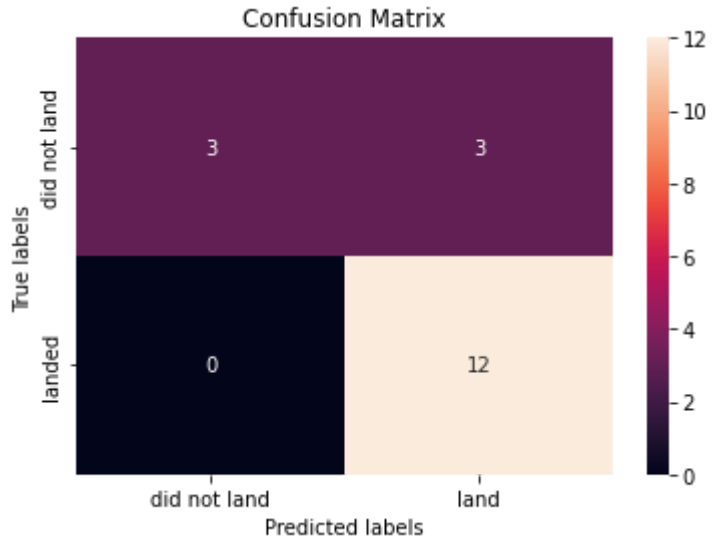
```
print("tuned hpyerparameters :(best parameters) ",svm_cv.best_params_)
print("accuracy :",svm_cv.best_score_)
```

```
tuned hpyerparameters :(best parameters)  {'C': 1.0, 'gamma': 0.03162277660168379, 'k
accuracy : 0.8482142857142856
```

```
#TASK 7
#Calculate the accuracy on the test data using the method score:

svm_cv.score(X_test, Y_test)
```

```
0.8333333333333334
```

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




```
#TASK 8
#Create a decision tree classifier object then create a GridSearchCV object tree_cv with c

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()


# Instantiate the GridSearchCV object: tree_cv
tree_cv = GridSearchCV(tree, parameters, cv=10)

# Fit it to the data
tree_cv.fit(X_train, Y_train)
```

```
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],
```

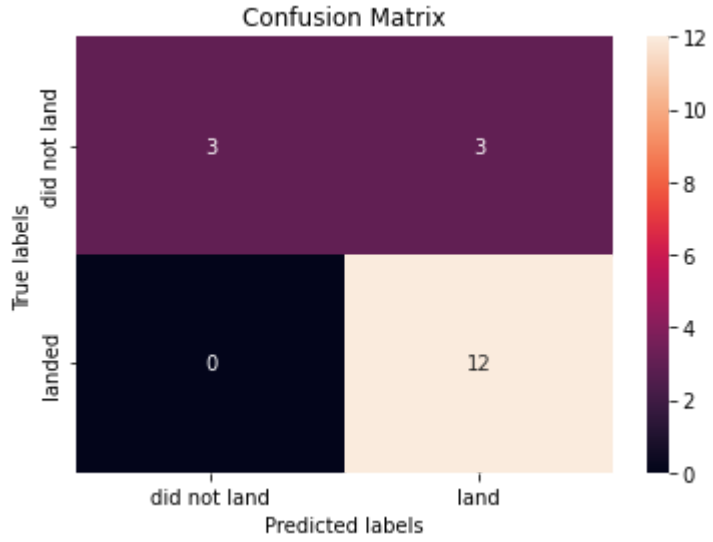
```
                                'min_samples_split': [2, 5, 10],
                                'splitter': ['best', 'random']})
```

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

```
tuned hpyerparameters :(best parameters)  {'criterion': 'gini', 'max_depth': 6, 'max_
accuracy : 0.875
```

```
#TASK 9
#Calculate the accuracy of tree_cv on the test data using the method score:

print(tree_cv.score(X_test, Y_test))
#We can plot the confusion matrix

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

```
0.7222222222222222
```




```
#task-10
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()
# Instantiate the GridSearchCV object: knn_cv
knn_cv = GridSearchCV(KNN, parameters, cv=10)

# Fit it to the data
knn_cv.fit(X_train, Y_train)
```

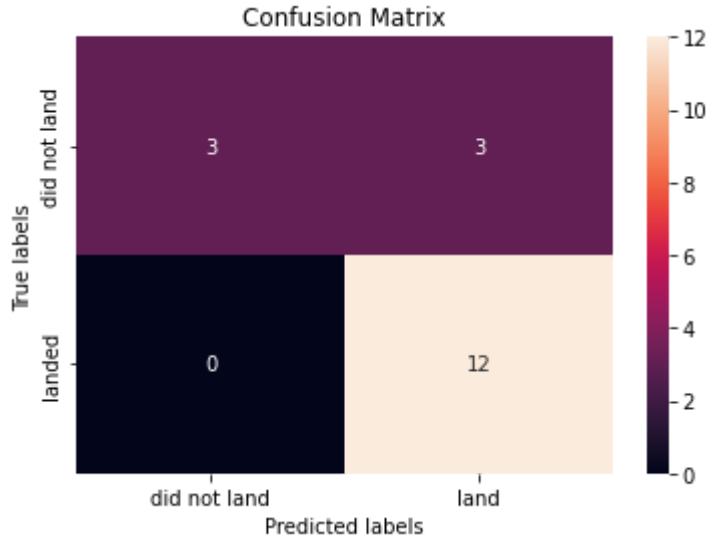
```
GridSearchCV(cv=10, estimator=KNeighborsClassifier(),
             param_grid={'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],
                         'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
                         'p': [1, 2]})
```

```
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
accuracy : 0.8482142857142858
```

```
#TASK 11
#Calculate the accuracy of tree_cv on the test data using the method score:

print(knn_cv.score(X_test, Y_test))
#We can plot the confusion matrix

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

```
0.8333333333333334
```




```
#TASK 12
#Find the method performs best:

predictors = [knn_cv, svm_cv, logreg_cv, tree_cv]
best_predictor = ""
best_result = 0
for predictor in predictors:

    predictor.score(X_test, Y_test)
```

✓ 0s completed at 11:38 PM