INTRODUCTION

1. **Exploratory Data Analysis (EDA)**: Analyzing and understanding the dataset to identify patterns, trends, and key features related to the landing outcome.
2. **Data Preparation**: Preprocessing the data, standardizing it, and splitting it into training and testing sets.
3. **Model Building**: Training and evaluating multiple machine learning models, including `Logistic Regression`, `Support Vector Machines`, `Decision Trees`, and `K-Nearest Neighbors`.
4. **Model Evaluation**: Assessing the models' performance choosing the best accuracy.

We will now perform some Exploratory Data Analysis (EDA) to find some patterns in the data

1. SQL Queries

Objectives

- Understand the Spacex DataSet
- Load the dataset into the corresponding table in a Db2 database
- Execute SQL queries to answer assignment questions

Display the names of the unique launch sites in the space mission

```
select distinct Launch_Site FROM SPACEXTABLE
```

```
In [36]:
# Print the result for the query
print_querry[1]
```

Out[36]:

	LAUNCH_SITE
0	CCAFS LC-40
1	CCAFS SLC-40
2	KSC LC-39A
3	VAFB SLC-4E

Query 2: Display 5 records where launch sites begin with the string

```
select * from SPACEXTABLE where Launch_Site LIKE 'CCA%' limit 5
```

In [37]:

```
# Print the result for the query
print_querry[2]
```

Out[37]:

	DATE	TIME__UTC_	BOOSTER_VERSION	LAUNCH_SITE	PAYLOAD	PAYLOAD_MASS__KG_	ORBIT	CUSTOMER	MISSION_OUTCOME	LANDING__OUTCOME
0	2010-06-04	18:45:00	F9 v1.0 B0003	CCAFS LC-40	Dragon Spacecraft Qualification Unit	0	LEO	SpaceX	Success	Failure (parachute)
1	2010-12-08	15:43:00	F9 v1.0 B0004	CCAFS LC-40	Dragon demo flight C1, two CubeSats, barrel of...	0	LEO (ISS)	NASA (COTS) NRO	Success	Failure (parachute)
2	2012-05-22	07:44:00	F9 v1.0 B0005	CCAFS LC-40	Dragon demo flight C2	525	LEO (ISS)	NASA (COTS)	Success	No attempt
3	2012-10-08	00:35:00	F9 v1.0 B0006	CCAFS LC-40	SpaceX CRS-1	500	LEO (ISS)	NASA (CRS)	Success	No attempt
4	2013-03-01	15:10:00	F9 v1.0 B0007	CCAFS LC-40	SpaceX CRS-2	677	LEO (ISS)	NASA (CRS)	Success	No attempt

Query 3: Display the total payload mass carried by boosters launched by NASA (CRS)

```
select sum(PAYLOAD_MASS__KG_) from SPACEXTABLE where customer = 'NASA (CRS)'
```

```
In [38]:
# Print the result for the query
print_querry[3]
```

Out[38]:

	1
0	45596

Query 4: Display mass carried by booster version F9 v1.1

```
select avg(PAYLOAD_MASS__KG_) from SPACEXTABLE where Booster_Version = 'F9 v1.1'
```

```
In [39]:
# Print the result for the query
print_querry[4]
```

Out[39]:

	1
0	2928

Query 5: List the date when the first succesful landing outcome in ground pad was acheived

```
select min(Date) from SPACEXTABLE where Landing_Outcome = 'Success (ground pad)'
```

```
In [40]:
# Print the result for the query
print_querry[5]
```

Out[40]:

	1
0	2015-12-22

Query 6: List the names of the boosters which have success in drone ship and have payload mass greater than 4000 but less than 6000

```
select distinct Booster_Version from SPACEXTABLE where Landing_Outcome = 'Success (drone ship)' and PAYLOAD_MASS__KG_ > 4000 and PAYLOAD_MASS__KG_ < 6000
```

In [41]:

```
# Print the result for the query
print_querry[6]
```

Out[41]:

	BOOSTER_VERSION
0	F9 FT B1021.2
1	F9 FT B1031.2
2	F9 FT B1022
3	F9 FT B1026

Query 7: List the total number of successful and failure mission outcomes

```
select distinct Mission_Outcome, count(*) from SPACEXTABLE group by Mission_Outcome
```

In [42]:

```
# Print the result for the query
print_querry[7]
```

Out[42]:

	MISSION_OUTCOME	2
0	Failure (in flight)	1
1	Success	99
2	Success (payload status unclear)	1

Query 8: List the names of the booster_versions which have carried the maximum payload mass. Use a subquery

```
select Booster_Version from SPACEXTABLE where PAYLOAD_MASS__KG_ = (select max(PAYLOAD_MASS__KG_)from SPACEXTABLE)
```

In [43]:

```
# Print the result for the query
print_querry[8]
```

Out[43]:

	BOOSTER_VERSION
0	F9 B5 B1048.4
1	F9 B5 B1049.4
2	F9 B5 B1051.3
3	F9 B5 B1056.4
4	F9 B5 B1048.5
5	F9 B5 B1051.4
6	F9 B5 B1049.5
7	F9 B5 B1060.2
8	F9 B5 B1058.3
9	F9 B5 B1051.6

	BOOSTER_VERSION
10	F9 B5 B1060.3
11	F9 B5 B1049.7

Query 9: List the records which will display the month names, failure landing_outcomes in drone ship ,booster versions, launch_site for the months in year 2015.

```
select substr(Date, 6,2) as Month, Landing_Outcome, Booster_Version, Launch_Site
from SPACEXTABLE where Landing_Outcome = 'Failure (drone ship)' and
substr(Date,0,5) = '2015'
```

or

```
select month(Date) as Month, Landing_Outcome, Booster_Version, Launch_Site from
SPACEXTABLE where Landing_Outcome = 'Failure (drone ship)' and year(Date) =
'2015'
```

```
In [44]:
# Print the result for the query
print_querry[9]
```

Out[44]:

	MONTH	LANDING__OUTCOME	BOOSTER_VERSION	LAUNCH_SITE
0	1	Failure (drone ship)	F9 v1.1 B1012	CCAFS LC-40
1	4	Failure (drone ship)	F9 v1.1 B1015	CCAFS LC-40

Query 10: Rank the count of landing outcomes (such as Failure (drone ship) or Success (ground pad)) between the date 2010-06-04 and 2017-03-20, in descending order.

```
select Landing_Outcome, count(*) as 'Count' from SPACEXTABLE where Date between
'2010-06-04' and '2017-03-20' group by Landing_Outcome order by Count desc
```

```
In [45]:
# Print the result for the query
print_querry[10]
```

Out[45]:

	LANDING__OUTCOME	COUNT
0	No attempt	10
1	Failure (drone ship)	5
2	Success (drone ship)	5
3	Controlled (ocean)	3
4	Success (ground pad)	3
5	Failure (parachute)	2
6	Uncontrolled (ocean)	2
7	Precluded (drone ship)	1

Data Visualization

Objectives

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

- Exploratory Data Analysis
- Preparing Data Feature Engineering

In [46]:
```
import matplotlib.pyplot as plt
import seaborn as sns
```

In [47]:
```
df=pd.read_csv("https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/dataset_part_2.csv")
```

In [48]:

```
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()
```

Flight Number vs Launch Site

```
In [49]:
# Plot a scatter point chart with x axis to be Flight Number and y axis to be the lau
nch site, and hue to be the class value
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()
```

Payload vs Launch Site

```
In [50]:
# Plot a scatter point chart with x axis to be Pay Load Mass (kg) and y axis to be th
e launch site, and hue to be the class value
sns.catplot(y='LaunchSite', x='PayloadMass', hue='Class', data=df, aspect=5)
plt.xlabel("Pay Load Mass (Kg)",fontsize=20)
plt.ylabel("Launch Site)",fontsize=20)
plt.show()
```

Success Rate vs Orbit Type

```
.In [51]:

orbit_success =
df.groupby(['Orbit'])['Class'].aggregate(np.average).reset_index().sort_values(['Clas
s','Orbit'], ascending=False)

orbit_success['Class'] = np.round(orbit_success['Class']*100, 2)
sns.barplot(x='Orbit', y='Class', data=orbit_success)
plt.ylabel("Success Rate (%)",fontsize=20)
plt.xlabel("Orbit Type",fontsize=20)
plt.show()
```

.FlightNumber vs Orbit Type

```
In [52]:
# Plot a scatter point chart with x axis to be FlightNumber and y axis to be the Orbi
t, and hue to be the class value
sns.catplot(x='FlightNumber', y='Orbit', hue='Class', data=df, aspect=5)
plt.xlabel("Flight Number",fontsize=20)
plt.ylabel("Orbit Type",fontsize=20)
plt.show()
```

Payload vs Orbit Type

```
In [53]:
# Plot a scatter point chart with x axis to be Payload and y axis to be the Orbit, an
d hue to be the class value
sns.catplot(x='PayloadMass', y='Orbit', hue='Class', data=df, aspect=5)
```

```
plt.xlabel("Pay Load Mass (Kg)",fontsize=20)
plt.ylabel("Orbit Type",fontsize=20)
plt.show()
```

Launch Success Yearly Trend

```
In [54]:
# Plot a line chart with x axis to be the extracted year and y axis to be the success
rate

df['Year'] = [i.split("-")[0] for i in df["Date"]]
yearly_success = df.groupby(['Year'])['Class'].aggregate(np.average).reset_index().so
rt_values('Year')
yearly_success['Class'] = np.round(yearly_success['Class']*100, 2)

sns.lineplot(x='Year',y='Class', data=yearly_success)
plt.xlabel("Year",fontsize=20)
plt.ylabel("Success Rate (%)",fontsize=20)
plt.show()
```

.

Interactive map

Objectives

- Mark all launch sites on a Folium map

```
In [55]:
!pip3 install folium
!pip3 install wget

import folium
import wget
# Import folium MarkerCluster plugin
from folium.plugins import MarkerCluster
# Import folium MousePosition plugin
from folium.plugins import MousePosition
# Import folium DivIcon plugin
from folium.features import DivIcon
```

```
Requirement already satisfied: folium in /opt/conda/lib/python3.10/site-packages
(0.14.0)
Requirement already satisfied: branca>=0.6.0 in /opt/conda/lib/python3.10/site-pa
ckages (from folium) (0.6.0)
Requirement already satisfied: jinja2>=2.9 in /opt/conda/lib/python3.10/site-pack
ages (from folium) (3.1.2)
Requirement already satisfied: numpy in /opt/conda/lib/python3.10/site-packages (
from folium) (1.23.5)
Requirement already satisfied: requests in /opt/conda/lib/python3.10/site-package
s (from folium) (2.31.0)
Requirement already satisfied: MarkupSafe>=2.0 in /opt/conda/lib/python3.10/site-
packages (from jinja2>=2.9->folium) (2.1.3)
Requirement already satisfied: charset-normalizer<4,>=2 in /opt/conda/lib/python3
.10/site-packages (from requests->folium) (3.1.0)
```

```
Requirement already satisfied: idna<4,>=2.5 in /opt/conda/lib/python3.10/site-pac
kages (from requests->folium) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in /opt/conda/lib/python3.10/si
te-packages (from requests->folium) (1.26.15)
Requirement already satisfied: certifi>=2017.4.17 in /opt/conda/lib/python3.10/si
te-packages (from requests->folium) (2023.7.22)
Collecting wget
  Downloading wget-3.2.zip (10 kB)
  Preparing metadata (setup.py) ... done
Building wheels for collected packages: wget
  Building wheel for wget (setup.py) ... - done
  Created wheel for wget: filename=wget-3.2-py3-none-any.whl size=9657 sha256=067
c97b79c775427484b7516255f87d9c0744714ee710f964f12b9b652e0d05f
  Stored in directory: /root/.cache/pip/wheels/8b/f1/7f/5c94f0a7a505ca1c81cd1d920
8ae2064675d97582078e6c769
Successfully built wget
Installing collected packages: wget
Successfully installed wget-3.2
```

Marking all launch sites on a Folium map

In [56]:

```
spacex_csv_file = wget.download('https://cf-courses-data.s3.us.cloud-object-storage.a
ppdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/spacex_launch_geo.csv')
spacex_df=pd.read_csv(spacex_csv_file)
```

In [57]:

```
# Select relevant sub-columns: `Launch Site`, `Lat(Latitude)`, `Long(Longitude)`, `cl
ass`
spacex_df = spacex_df[['Launch Site', 'Lat', 'Long', 'class']]
launch_sites_df = spacex_df.groupby(['Launch Site'], as_index=False).first()
launch_sites_df = launch_sites_df[['Launch Site', 'Lat', 'Long']]
launch_sites_df
```

Out[57]:

	Launch Site	Lat	Long
0	CCAFS LC-40	28.562302	-80.577356
1	CCAFS SLC-40	28.563197	-80.576820
2	KSC LC-39A	28.573255	-80.646895
3	VAFB SLC-4E	34.632834	-120.610745

In [58]:
```
# Create a folium `Map` object, with an initial center location to be NASA Johnson Sp
ace Center at Houston, Texas
nasa_coordinate = [29.559684888503615, -95.0830971930759]
site_map = folium.Map(location=nasa_coordinate, zoom_start=10)
```

In [59]:
```
# Create a blue circle at NASA Johnson Space Center's coordinate with a popup label s
howing its name
circle = folium.Circle(nasa_coordinate, radius=1000, color='#d35400', fill=True).add_
child(folium.Popup('NASA Johnson Space Center'))

# Create a blue circle at NASA Johnson Space Center's coordinate with a icon showing
its name
marker = folium.map.Marker(
    nasa_coordinate,
    # Create an icon as a text label
    icon=DivIcon(
        icon_size=(20,20),
        icon_anchor=(0,0),
        html='<div style="font-size: 12; color:#d35400;"><b>%s</b></div>' % 'NASA JSC
',
        )
    )
site_map.add_child(circle)
site_map.add_child(marker)
```

Out[59]:

In [60]:
```
Circle object based on its coordinate (Lat, Long) values.
for i, row in launch_sites_df.iterrows():
    coordinate = [row['Lat'], row['Long']]
    # In addition, add Launch site name as a popup label
    circle = folium.Circle(coordinate, radius=1000, color='#d35400', fill=True).add_c
hild(folium.Popup(row['Launch Site']))
    marker = folium.map.Marker(coordinate, icon=DivIcon(icon_size=(20,20),icon_anchor
=(0,0), html='<div style="font-size: 12; color:#d35400;"><b>%s</b></div>' % row['Laun
ch Site'], ))
    site_map.add_child(circle)
    site_map.add_child(marker)
site_map
```

Out[60]:

NB: If a launch was successful (`class=1`), then we use a green marker and if a launch was failed, we use a red marker (`class=0`)

In [61]:
```
# Create a `MarkerCluster` object
marker_cluster = MarkerCluster()
```

In [62]:
```
# Assign color to launch outcome
def assign_marker_color(launch_outcome):
    # If class=1, marker_color value will be green
    if launch_outcome == 1:
```

```
        return 'green'
    # If class=0, marker_color value will be red
    else:
        return 'red'

spacex_df['marker_color'] = spacex_df['class'].apply(assign_marker_color)
spacex_df.tail(10)
```

Out[62]:

	Launch Site	Lat	Long	class	marker_color
46	KSC LC-39A	28.573255	-80.646895	1	green
47	KSC LC-39A	28.573255	-80.646895	1	green
48	KSC LC-39A	28.573255	-80.646895	1	green
49	CCAFS SLC-40	28.563197	-80.576820	1	green
50	CCAFS SLC-40	28.563197	-80.576820	1	green
51	CCAFS SLC-40	28.563197	-80.576820	0	red
52	CCAFS SLC-40	28.563197	-80.576820	0	red
53	CCAFS SLC-40	28.563197	-80.576820	0	red
54	CCAFS SLC-40	28.563197	-80.576820	1	green
55	CCAFS SLC-40	28.563197	-80.576820	0	red

In [63]:

```
# Add marker_cluster to current site_map
```

```
site_map.add_child(marker_cluster)

# Create Marker object with coordinate and customize icon property to indicate if thi
s launch was successed or failed
for i, row in spacex_df.iterrows():
    coordinate = [row['Lat'], row['Long']]
    marker = folium.map.Marker(coordinate, icon=folium.Icon(color='white',icon_color=
row['marker_color']))
    marker_cluster.add_child(marker)

site_map
```

In [64]:

```
# Add Mouse Position to get the coordinate (Lat, Long) for a mouse over on the map
formatter = "function(num) {return L.Util.formatNum(num, 5);};"
mouse_position = MousePosition(
    position='topright',
    separator=' Long: ',
    empty_string='NaN',
    lng_first=False,
    num_digits=20,
    prefix='Lat:',
    lat_formatter=formatter,
    lng_formatter=formatter,
)

site_map.add_child(mouse_position)
site_map
```

```
# Calculate the distance between two points on the map based on their `Lat` and `Long
` values

from math import sin, cos, sqrt, atan2, radians

def calculate_distance(lat1, lon1, lat2, lon2):
    # approximate radius of earth in km
    R = 6373.0

    lat1 = radians(lat1)
    lon1 = radians(lon1)
    lat2 = radians(lat2)
    lon2 = radians(lon2)

    dlon = lon2 - lon1
    dlat = lat2 - lat1

    a = sin(dlat / 2)**2 + cos(lat1) * cos(lat2) * sin(dlon / 2)**2
    c = 2 * atan2(sqrt(a), sqrt(1 - a))

    distance = R * c
    return distance
```

In [66]:

```
# Mark down a point on the closest coastline using MousePosition and calculate the di
stance between the coastline point and the launch site:
```

```
launch_site_lat = launch_sites_df['Lat'][1]
launch_site_lon = launch_sites_df['Long'][1]

coastline_lat = 28.56362
coastline_lon = -80.56802

distance_coastline = calculate_distance(launch_site_lat, launch_site_lon, coastline_l
at, coastline_lon)

distance_coastline
```

Out[66]:

```
0.8609769661763733
```

In [67]:

```
# Create and add a folium.Marker on the selected closest coastline point on the map
# Display the distance between coastline point and launch site using the icon propert
y

coordinate = [coastline_lat, coastline_lon]

distance_marker = folium.Marker(
   coordinate,
   icon=DivIcon(
       icon_size=(20,20),
       icon_anchor=(0,0),
       html='<div style="font-size: 12; color:#d35400;"><b>%s</b></div>' % "{:10.2f}
KM".format(distance_coastline),
    )
)

site_map.add_child(distance_marker)

# Create a `folium.PolyLine` object using the coastline coordinates and launch site c
oordinate
# lines=folium.PolyLine(locations=coordinates, weight=1)

line=folium.PolyLine(locations=[[launch_site_lat,launch_site_lon],[coastline_lat,coas
tline_lon]], weight=1)
site_map.add_child(line)
```

Out[67]:

In [68]:

```
# Create a marker with distance to a closest city, railway, highway, etc.
# Draw a line between the marker to the launch site

closest_city = [28.0948, -80.6369]
closest_railway = [28.57203, -80.58525]
closest_highway = [28.56416, -80.57086]
```

In [69]:

```
# Closest City

distance_closest_city = calculate_distance(launch_site_lat, launch_site_lon, closest_
city[0], closest_city[1])
```

```
city_marker = folium.Marker(
   closest_city,
   icon=DivIcon(
       icon_size=(20,20),
       icon_anchor=(0,0),
       html='<div style="font-size: 12; color:#d35400;"><b>%s</b></div>' % "{:10.2f}
KM".format(distance_closest_city),
    )
)

city_line = folium.PolyLine(locations=[[launch_site_lat,launch_site_lon],closest_city
], weight=1)

site_map.add_child(city_marker)
site_map.add_child(city_line)
```

In [70]:

```
# Closest Railway
distance_closest_railway = calculate_distance(launch_site_lat, launch_site_lon, close
st_railway[0], closest_railway[1])

railway_marker = folium.Marker(
   closest_railway,
   icon=DivIcon(
       icon_size=(20,20),
       icon_anchor=(0,0),
       html='<div style="font-size: 12; color:#d35400;"><b>%s</b></div>' % "{:10.2f}
KM".format(distance_closest_railway),
    )
)

railway_line = folium.PolyLine(locations=[[launch_site_lat,launch_site_lon],closest_r
ailway], weight=1)

site_map.add_child(railway_marker)
site_map.add_child(railway_line)
```

In [71]:

```
# Closest Highway
distance_closest_highway = calculate_distance(launch_site_lat, launch_site_lon, close
st_highway[0], closest_highway[1])

highway_marker = folium.Marker(
   closest_highway,
   icon=DivIcon(
       icon_size=(20,20),
       icon_anchor=(0,0),
       html='<div style="font-size: 12; color:#d35400;"><b>%s</b></div>' % "{:10.2f}
KM".format(distance_closest_highway),
    )
)

highway_line = folium.PolyLine(locations=[[launch_site_lat,launch_site_lon],closest_h
ighway], weight=1)
```

```
site_map.add_child(highway_marker)
site_map.add_child(highway_line)
```

Interactive Dashboard

```
!wget "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321E
N-SkillsNetwork/datasets/spacex_launch_dash.csv"
```

In [73]:

```
!pip install dash
```

In [74]:

```
# Import required libraries
import pandas as pd
import dash
from dash import html
from dash import dcc
from dash.dependencies import Input, Output
import plotly.express as px

# Read the airline data into pandas dataframe
spacex_df = pd.read_csv("spacex_launch_dash.csv")
spacex_df['Launch Outcome'] = ['Success' if c == 1 else 'Failure' for c in spacex_df[
'class']]
max_payload = spacex_df['Payload Mass (kg)'].max()
min_payload = spacex_df['Payload Mass (kg)'].min()

# Create a dash application
app = dash.Dash(__name__)

# Create an app layout and CSS style
app.layout = html.Div(children = [
    html.H1('SpaceX Launch Records Dashboard',
            style = {'textAlign': 'center', 'color': '#503D36', 'font-size': 40}),

    # TASK 1: Add a dropdown list to enable Launch Site selection
    # The default select value is for ALL sites
    dcc.Dropdown(id = 'site-dropdown',
                options = [
                            {'label': 'All Sites',
                                'value': 'ALL'},
                            {'label': 'CCAFS LC-40',
                                'value': 'CCAFS LC-40'},
                            {'label': 'CCAFS SLC-40',
                                'value': 'CCAFS SLC-40'},
                            {'label': 'KSC LC-39A',
                                'value': 'KSC LC-39A'},
                            {'label': 'VAFB SLC-4E',
                                'value': 'VAFB SLC-4E'}],
                value = 'ALL',
                placeholder = 'Launch Site:',
                searchable = True ),

    # TASK 2: Add a pie chart to show the total successful launches count for all sit
es
    # If a specific launch site was selected, show the Success vs. Failed counts for
the site
```

```
    html.Div(dcc.Graph(id = 'success-pie-chart')),
    html.Br(),
    html.P("Payload range (Kg):"),

    # TASK 3: Add a slider to select payload range
    # dcc.RangeSlider(id='payload-slider',...)
    dcc.RangeSlider(id = 'payload-slider',
                    min = 0, max = 10000, step = 1000,
                    marks = {0: '0',
                             1000: '1000',
                             2000: '2000',
                             3000: '3000',
                             4000: '4000',
                             5000: '5000',
                             6000: '6000',
                             7000: '7000',
                             8000: '8000',
                             9000: '9000',
                             10000: '10000'},
                    value = [min_payload, max_payload]),

    # TASK 4: Add a scatter chart to show the correlation between payload and launch
success
    html.Div(
        dcc.Graph(id = 'success-payload-scatter-chart')),
    ], style = {'height': '100vh'}
)

# TASK 2:
# Add a callback function for `site-dropdown` as input, `success-pie-chart` as output
@app.callback(Output(component_id = 'success-pie-chart', component_property = 'figure
'),
              Input(component_id = 'site-dropdown', component_property = 'value'))
def get_pie_chart(entered_site):
    filtered_df = spacex_df
    if entered_site == 'ALL':
        fig = px.pie(filtered_df, values = 'class',
                     names = 'Launch Site',
                     title = 'SpaceX Launch Site Success Distribution (All Sites)')
        return fig
    else:
        # return the outcomes piechart for a selected site
        filtered_df = spacex_df[spacex_df['Launch Site'] == entered_site]
        filtered_df = filtered_df.groupby(['Launch Site', 'class']).size().reset_inde
x(name = 'class_count')
        fig = px.pie(filtered_df, values = 'class_count', names = filtered_df['class'
].map({1: "Success", 0: "Failure"}),
                     title = f"SpaceX Success Rate of {entered_site} Launch Site")
        fig.update_traces(marker = dict(colors=['red', 'green']))
        return fig

# TASK 4:
# Add a callback function for `site-dropdown` and `payload-slider` as inputs, `succes
s-payload-scatter-chart` as output
```

```
@app.callback(Output(component_id = 'success-payload-scatter-chart', component_proper
ty = 'figure'),
              [Input(component_id = 'site-dropdown', component_property = 'value'),
               Input(component_id = 'payload-slider', component_property = 'value')])
def get_scatter_chart(entered_site, payload):
    filtered_df = spacex_df[spacex_df['Payload Mass (kg)'].between(
        payload[0], payload[1])]
    if entered_site == 'ALL':
        fig = px.scatter(filtered_df, x = 'Payload Mass (kg)', y = 'Launch Outcome',
                         color = 'Booster Version Category', title = 'Success vs. Fai
lure for Payload Mass and Launch Sites (All Sites)')
        return fig
    else:
        fig = px.scatter(filtered_df[filtered_df['Launch Site'] == entered_site], x =
'Payload Mass (kg)', y = 'Launch Outcome',
                         color = 'Booster Version Category', title = f"Success vs. Fa
ilure for Payload Mass and Launch Site {entered_site}")
        return fig
```

PREDICTIVE ANALYSIS

In [75]:
```
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
```

In [76]:
```
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'])
```

In [77]:
```
data = pd.read_csv("https://cf-courses-data.s3.us.cloud-object-storage.appdomain.clou
d/IBM-DS0321EN-SkillsNetwork/datasets/dataset_part_2.csv")

data.head()
```

Out[77]:

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPad	Block	ReusedCount	Serial	Longitude	Latitude	Class
0	1	2010-06-04	Falcon 9	6104.959412	LEO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B0003	-80.577366	28.561857	0
1	2	2012-05-22	Falcon 9	525.000000	LEO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B0005	-80.577366	28.561857	0
2	3	2013-03-01	Falcon 9	677.000000	ISS	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B0007	-80.577366	28.561857	0
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	34.632093	0
4	5	2013-12-	Falcon 9	3170.000000	GTO	CCAFS SL	None None	1	False	False	False	NaN	1.0	0	B1004	-80.577366	28.561857	0

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPad	Block	ReusedCount	Serial	Longitude	Latitude	Class
		03				C 40												

In [78]:

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

X.head()
```

Out[78]:

	FlightNumber	PayloadMass	Flights	Block	ReusedCount	Orbit_ES-L1	Orbit_GEO	Orbit_GTO	Orbit_HEO	Orbit_ISS	...	Serial_B1058	Serial_B1059	Serial_B1060	Serial_B1062	GridFins_False	GridFins_True	Reused_False	Reused_True	Legs_False	Legs_True
0	1.0	6104.959412	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0
1	2.0	525.000000	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0
2	3.0	677.000000	1.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	...	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0

	FlightNumber	PayloadMass	Flights	Block	ReusedCount	Orbit_ES-L1	Orbit_GEO	Orbit_GTO	Orbit_HEO	Orbit_ISS	...	Serial_B1058	Serial_B1059	Serial_B1060	Serial_B1062	GridFins_False	GridFins_True	Reused_False	Reused_True	Legs_False	Legs_True
3	4.0	500.000000	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0
4	5.0	3170.0000000	1.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	...	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0

5 rows × 83 columns

In [79]:

```
# Create a NumPy array from the column Class
y = data['Class'].to_numpy()
y
```

Out[79]:

```
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])
```

In [80]:

```
# Standardize the data in X
transform = preprocessing.StandardScaler()
X = transform.fit_transform(X)
X
```

Out[80]:

```
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]])
```

In [81]:

```
# Split the data X and Y into training and test data
X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.2, random_stat
e=2)
y_test.shape
```

Out[81]:

```
(18,)
```

Logistic Regression Model (LR)

In [82]:

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


lr=LogisticRegression()
logreg_cv = GridSearchCV(lr, parameters, cv=10).fit(X_train, y_train)
```

In [83]:

```
print("LR Tuned Hyperparameters (best parameters):",logreg_cv.best_params_)
print("LR Train Accuracy:",logreg_cv.best_score_)
```

```
LR Tuned Hyperparameters (best parameters): {'C': 0.01, 'penalty': 'l2', 'solver'
: 'lbfgs'}
LR Train Accuracy: 0.8464285714285713
```

In [84]:

```
logreg_accuracy = logreg_cv.score(X_test, y_test)
print(logreg_accuracy)
```

```
0.8333333333333334
```

Plotting confusion matrix

In [85]:

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

Support Vector Machine Model (SVM)

In [86]:

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

grid_search = GridSearchCV(svm, parameters, cv=10)
svm_cv = grid_search.fit(X_train, y_train)
```

In [87]:

```
print("SVM Tuned Hyperparameters (best parameters):",svm_cv.best_params_)
print("SVM Train Accuracy:",svm_cv.best_score_)
```

```
SVM Tuned Hyperparameters (best parameters): {'C': 1.0, 'gamma': 0.03162277660168
379, 'kernel': 'sigmoid'}
SVM Train Accuracy: 0.8482142857142856
```

In [88]:

```
svm_accuracy = svm_cv.score(X_test, y_test)
print("SVM Test Accuracy:",svm_accuracy)
```

```
SVM Test Accuracy: 0.8333333333333334
```

SVM Confusion Matrix

In [89]:

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

Decision Tree Model (DT)

In [90]:

```
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()

grid_search = GridSearchCV(tree, parameters, cv=10)
tree_cv = grid_search.fit(X_train, y_train)
```

In [91]:

```
print("DT Tuned Hyperparameters (best parameters):",tree_cv.best_params_)
print("DT Train Accuracy:",tree_cv.best_score_)
```

```
DT Tuned Hyperparameters (best parameters): {'criterion': 'entropy', 'max_depth':
6, 'max_features': 'sqrt', 'min_samples_leaf': 1, 'min_samples_split': 10, 'split
ter': 'random'}
DT Train Accuracy: 0.8892857142857145
```

In [92]:

```
tree_accuracy = tree_cv.score(X_test, y_test)
print("DT Test Accuracy:",tree_accuracy)
```

```
DT Test Accuracy: 0.8333333333333334
```

DT Confusion Matrix

In [93]:

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

K-Nearest Neighbors Model (KNN)

In [94]:

```
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()

grid_search = GridSearchCV(KNN, parameters, cv=10)
knn_cv = grid_search.fit(X_train, y_train)
```

In [95]:

```
print("KNN Tuned Hyperparameters (best parameters):",knn_cv.best_params_)
print("KNN Train Accuracy:",knn_cv.best_score_)
```

```
KNN Tuned Hyperparameters (best parameters): {'algorithm': 'auto', 'n_neighbors':
10, 'p': 1}
KNN Train Accuracy: 0.8482142857142858
```

In [96]:

```
knn_accuracy = knn_cv.score(X_test, y_test)
print("KNN Test Accuracy:",knn_accuracy)
```

```
KNN Test Accuracy: 0.8333333333333334
```

Plot KNN Confusion Matrix

In [97]:

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