**Libraries etc.**

!pip3 install beautifulsoup4

!pip3 install requests

!pip install sqlalchemy==1.3.9

!pip install -q pandas==1.1.5

!pip install numpy

!pip install pandas

!pip install seaborn

!pip install folium

!pip install SQLAlchemy

# 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 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

# Datetime is a library that allows us to represent dates

import datetime

import sys

from bs4 import BeautifulSoup

import re

import unicodedata

import csv, sqlite3

import matplotlib.pyplot as plt

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

%load\_ext sql

%sql sqlite:///my\_data1.db

%sql create table SPACEXTABLE as select \* from SPACEXTBL where Date is not null

# 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)

pip install folium –upgrade

**Import data**

# Takes the dataset and uses the rocket column to call the API and append the data to the list

def getBoosterVersion(data):

for x in data['rocket']:

if x:

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 the list

def getLaunchSite(data):

for x in data['launchpad']:

if x:

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 the lists

def getPayloadData(data):

for load in data['payloads']:

if load:

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 lists

def getCoreData(data):

for core in data['cores']:

if core['core'] != None:

response = requests.get("https://api.spacexdata.com/v4/cores/"+core['core']).json()

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)

# First Stage Landing Prediction

df1=pd.read\_csv("https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/dataset\_part\_1.csv")

df1.head(5)

# First Stage Landing Prediction

df1=pd.read\_csv("https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/dataset\_part\_1.csv")

df1.head(5)

# For SQL

df2 = pd.read\_csv("https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/labs/module\_2/data/Spacex.csv")

df2.head(20)

# Landing Locations

df3 = pd.read\_csv("https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/dataset\_part\_2.csv")

df3.head(5)

# Folium nonsense

df4 = pd.read\_csv("https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/spacex\_launch\_geo.csv")

df4.head(25)

**Plots**

# Show a scatter plot of Flight Number vs. Launch Site

plt.figure(figsize=(10, 6))

plt.scatter(df4['Flight Number'], df4['Launch Site'], marker='o', color='blue')

plt.title('Flight Number vs. Launch Site')

plt.xlabel('Flight Number')

plt.ylabel('Launch Site')

plt.grid(True)

plt.show()

# Show a scatter plot of Payload vs. Launch Site

plt.figure(figsize=(10, 6))

plt.scatter(df4['Payload'], df4['Launch Site'], marker='o', color='blue')

plt.title('Flight Number vs. Launch Site')

plt.xlabel('Flight Number')

plt.ylabel('Launch Site')

plt.grid(True)

plt.show()

# Show a bar chart for the success rate of each orbit type

# Calculate success rate for each orbit type

orbit\_success\_rate = df2.groupby('Orbit')['Mission\_Outcome'].mean() \* 100 # Calculate mean success rate in each orbit type and convert to percentage

# Create a bar chart for the success rate of each orbit type

plt.figure(figsize=(10, 6))

orbit\_success\_rate.plot(kind='bar', color='green')

plt.title('Success Rate of Each Orbit Type')

plt.xlabel('Orbit Type')

plt.ylabel('Success Rate (%)')

plt.xticks(rotation=45, ha='right') # Rotate x-axis labels for better readability

plt.grid(axis='y')

plt.tight\_layout()

plt.show()

# Show a scatter point of Flight number vs. Orbit type

plt.figure(figsize=(10, 6))

plt.scatter(df4['Flight Number'], df4['Orbit'], marker='o', color='orange')

plt.title('Flight Number vs. Orbit Type')

plt.xlabel('Flight Number')

plt.ylabel('Orbit Type')

plt.grid(True)

plt.show()

# Show a scatter point of payload vs. orbit type

plt.figure(figsize=(10, 6))

plt.scatter(df4['Payload'], df4['Orbit'], marker='o', color='purple')

plt.title('Payload vs. Orbit Type')

plt.xlabel('Payload')

plt.ylabel('Orbit Type')

plt.grid(True)

plt.show()

# Show a line chart of yearly average success rate

# Convert the 'Date' column to datetime format

df2['Date'] = pd.to\_datetime(df2['Date'])

# Extract the year from the 'Date' column

df2['Year'] = df2['Date'].dt.year

# Group the DataFrame by year and calculate the average success rate for each year

yearly\_avg\_success\_rate = df2.groupby('Year')['Mission\_Outcome'].mean()

# Plot the yearly average success rate as a line chart

plt.figure(figsize=(10, 6))

yearly\_avg\_success\_rate.plot(kind='line', marker='o', color='red')

plt.title('Yearly Average Success Rate')

plt.xlabel('Year')

plt.ylabel('Average Success Rate')

plt.grid(True)

plt.show()

# Find the names of the unique launch sites

# Get the unique launch site names

unique\_launch\_sites = df4['Launch Site'].unique()

# Print the unique launch site names

print(unique\_launch\_sites)

# Find 5 records where launch sites begin with 'CCA'

cca\_launch\_sites = df4[df4['Launch Site'].str.startswith('CCA')].head(5)

# Print the records

print(cca\_launch\_sites)

# Calculate the total payload carried by boosters from NASA

# Filter the DataFrame to include only the records where the 'Customer' column contains 'NASA'

nasa\_payload = df2[df2['Customer'].str.contains('NASA', case=False)]

# Calculate the total payload carried by boosters from NASA

total\_nasa\_payload = nasa\_payload['PAYLOAD\_MASS\_\_KG\_'].sum()

# Print the total payload carried by boosters from NASA

print("Total payload carried by boosters from NASA:", total\_nasa\_payload, "kg")

# Calculate the average payload mass carried by booster version F9 v1.1

# Filter the DataFrame to include only the records where the 'Booster Version' column is 'F9 v1.1'

f9\_v11\_payload = df4[df4['Booster Version'] == 'F9 v1.1']

# Calculate the average payload mass carried by booster version F9 v1.1

average\_payload\_f9\_v11 = f9\_v11\_payload['Payload Mass (kg)'].mean()

# Print the average payload mass carried by booster version F9 v1.1

print("Average payload mass carried by booster version F9 v1.1:", average\_payload\_f9\_v11, "kg")

# Find the dates of the first successful landing outcome on ground pad

# Filter the DataFrame to include only the records where the 'LandingPad' column is not null and the 'LandingOutcome' column is 'Success (ground pad)'

successful\_ground\_landing = df4[(df4['Landing Outcome'] == 'Success (ground pad)')]

# Find the earliest date from the filtered records

earliest\_successful\_ground\_landing\_date = successful\_ground\_landing['Date'].min()

# Print the date of the first successful landing outcome on a ground pad

print("Date of the first successful landing outcome on a ground pad:", earliest\_successful\_ground\_landing\_date)

# List the names of boosters which have successfully landed on drone ship and had payload mass greater than 4000 but less than 6000

# Filter the DataFrame to include only the records where the 'LandingType' column is 'ASDS' (drone ship) and the 'PayloadMass' column is between 4000 and 6000 (exclusive)

filtered\_boosters = df4[(df4['Landing Outcome'] == 'Success (drone ship)') & (df4['Payload Mass (kg)'] > 4000) & (df4['Payload Mass (kg)'] < 6000)]

# List the names of boosters which meet the criteria

boosters\_names = filtered\_boosters['Booster Version'].unique()

# Print the names of boosters

print("Names of boosters which have successfully landed on a drone ship and had payload mass greater than 4000 but less than 6000:")

print(boosters\_names)

# Calculate the total number of successful and failure mission outcomes

# Calculate the total number of successful and failure mission outcomes

success\_count = df2[df2['Mission\_Outcome'] == 'Success']['Mission\_Outcome'].count()

failure\_count = df2[df2['Mission\_Outcome'] != 'Success']['Mission\_Outcome'].count()

# Print the total number of successful and failure mission outcomes

print("Total number of successful mission outcomes:", success\_count)

print("Total number of failure mission outcomes:", failure\_count)

# List the names of the booster which have carried the maximum payload mass

# Find the maximum payload mass

max\_payload\_mass = df4['PAYLOAD\_MASS\_\_KG\_'].max()

# Filter the DataFrame to include only the records where the payload mass is equal to the maximum payload mass

max\_payload\_boosters = df4[df4['PAYLOAD\_MASS\_\_KG\_'] == max\_payload\_mass]

# List the names of the booster which have carried the maximum payload mass

booster\_names = max\_payload\_boosters['Booster Version'].unique()

# Print the names of the booster

print("Names of the booster which have carried the maximum payload mass:")

print(booster\_names)

# List the failed landing\_outcomes in drone ship, their booster versions, and launch site names for in year 2015

# Convert the 'Date' column to datetime format

df4['Date'] = pd.to\_datetime(df4['Date'])

# Extract the year from the 'Date' column

df4['Year'] = df4['Date'].dt.year

# Filter the DataFrame based on the conditions

failed\_landing = df4[(df4['Landing Outcome'] == 'Failure (drone ship)') & (df4['Year'] == 2015)]

# Select the columns of interest

failed\_landing\_info = failed\_landing[['Landing Outcome', 'Booster Version', 'Launch Site']]

# Print the filtered DataFrame

print(failed\_landing\_info)

# 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

# Convert the 'Date' column to datetime format

df4['Date'] = pd.to\_datetime(df4['Date'])

# Filter the DataFrame to include only the records between the specified dates

filtered\_df = df4[(df4['Date'] >= '2010-06-04') & (df4['Date'] <= '2017-03-20')]

# Group the filtered DataFrame by the 'LandingOutcome' column and count the occurrences of each landing outcome

landing\_outcome\_counts = filtered\_df['Landing Outcome'].value\_counts()

# Print the landing outcome counts in descending order

print("Ranking of landing outcomes between 2010-06-04 and 2017-03-20 (descending order):")

print(landing\_outcome\_counts)

# launch success count for all sites, in a piechart

# Filter the dataset for successful launches

successful\_launches = df2[df2['Mission\_Outcome'] == 'Success']

# Count the number of successful launches for each site

launch\_count\_by\_site = successful\_launches['Launch\_Site'].value\_counts()

# Plot the pie chart

plt.figure(figsize=(8, 8))

launch\_count\_by\_site.plot(kind='pie', autopct='%1.1f%%', startangle=90)

plt.title('Launch Success Count by Site')

plt.ylabel('')

plt.show()

# piechart for the launch site with highest launch success ratio

# Calculate the success ratio for each launch site

success\_ratio\_by\_site = df2.groupby('Launch\_Site')['Mission\_Outcome'].apply(lambda x: (x == 'Success').mean())

# Find the launch site with the highest success ratio

site\_with\_highest\_success\_ratio = success\_ratio\_by\_site.idxmax()

# Filter the dataset for the launch site with the highest success ratio

site\_data = df2[df2['Launch\_Site'] == site\_with\_highest\_success\_ratio]

# Count the number of successful launches and failures

success\_count = (site\_data['Mission\_Outcome'] == 'Success').sum()

failure\_count = (site\_data['Mission\_Outcome'] != 'Success').sum()

# Plot the pie chart

plt.figure(figsize=(8, 8))

plt.pie([success\_count, failure\_count], labels=['Success', 'Failure'], autopct='%1.1f%%', startangle=90)

plt.title(f'Launch Success Ratio for {site\_with\_highest\_success\_ratio}')

plt.show()

# Payload vs. Launch Outcome scatter plot for all sites, with different payload selected in the range slider

import pandas as pd

import plotly.express as px

# Load the dataset

df2 = pd.read\_csv("https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/labs/module\_2/data/Spacex.csv")

# Create a scatter plot with range slider

fig = px.scatter(df2, x='Payload', y='Mission\_Outcome', color='Launch\_Site',

title='Payload vs. Launch Outcome for all Sites',

range\_x=[df2['Payload'].min(), df2['Payload'].max()],

hover\_data=['Mission\_Outcome'])

fig.update\_layout(xaxis\_title='Payload', yaxis\_title='Launch Outcome')

fig.show()

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import confusion\_matrix

import matplotlib.pyplot as plt

import seaborn as sns

# Assuming you have already trained and evaluated your models and stored their performance metrics

# For demonstration, let's use Logistic Regression as the best performing model

# Split the data into features and target variable

X = df2[['Booster\_Version', 'Launch\_Site', 'Payload',

'PAYLOAD\_MASS\_\_KG\_', 'Orbit', 'Customer',

'Landing\_Outcome']]

y = df2['Mission\_Outcome']

# Dummy encoding categorical variables

X = pd.get\_dummies(X)

# Split the data into train and test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Train the model (assuming Logistic Regression)

model = LogisticRegression(max\_iter=1000)

model.fit(X\_train, y\_train)

# Predict on the test set

y\_pred = model.predict(X\_test)

# Generate the confusion matrix

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

# Plot the confusion matrix

plt.figure(figsize=(8, 6))

sns.heatmap(conf\_matrix, annot=True, fmt="d", cmap="Blues", cbar=False)

plt.xlabel('Predicted labels')

plt.ylabel('True labels')

plt.title('Confusion Matrix')

plt.show()