Skills Network

Space X Falcon 9 First Stage Landing Prediction

Assignment: Machine Learning Prediction

Estimated time needed: **60** minutes

Space X 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 Space X 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 space X 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.

Several examples of an unsuccessful landing are shown here:

Most unsuccessful landings are planed. Space X; performs a controlled landing in the oceans.

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

[1]:
```
import piplite
await piplite.install(['numpy'])
await piplite.install(['pandas'])
await piplite.install(['seaborn'])
```

We will import the following libraries for the lab

[2]:
```python
# 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 arr
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
# 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
```

This function is to plot the confusion matrix.

[3]:
```python
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'])
    plt.show()
```

Load the dataframe

Load the data

[4]:
```python
from js import fetch
import io

URL1 = "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/dataset_part_2.csv"
resp1 = await fetch(URL1)
text1 = io.BytesIO((await resp1.arrayBuffer()).to_py())
data = pd.read_csv(text1)
```

[5]:
```python
data.head()
```

[5]:

	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	2013-12-03	Falcon 9	3170.000000	GTO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B1004	-80.577366	28.561857	0

[6]:
```python
URL2 = 'https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/dataset_part_3.csv'
resp2 = await fetch(URL2)
text2 = io.BytesIO((await resp2.arrayBuffer()).to_py())
X = pd.read_csv(text2)
```

[7]:
```python
X.head(100)
```

[7]:

	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_Tru
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.
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.
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.
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.
4	5.0	3170.000000	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.
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	.
85	86.0	15400.000000	2.0	5.0	2.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	1.0	0.0	0.0	1.0	0.0	1.
86	87.0	15400.000000	3.0	5.0	2.0	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.
87	88.0	15400.000000	6.0	5.0	5.0	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.
88	89.0	15400.000000	3.0	5.0	2.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	1.0	0.0	0.0	1.0	0.0	1.
89	90.0	3681.000000	1.0	5.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	1.0	0.0	1.0	1.0	0.

90 rows × 83 columns

TASK 1

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']).

[8]:
```python
Y = data['Class'].to_numpy()
```

TASK 2

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

[9]:
```python
# students get this
transform = preprocessing.StandardScaler()

X= preprocessing.StandardScaler().fit(X).transform(X)
X[0:5]
```

[9]:
```
array([[-1.71291154e+00, -1.94814463e-16, -6.53912840e-01,
        -1.57589457e+00, -9.73440458e-01, -1.05999788e-01,
        -1.05999788e-01, -6.54653671e-01, -1.05999788e-01,
        -5.51677284e-01,  3.44342023e+00, -1.85695338e-01,
        -3.33333333e-01, -1.05999788e-01, -2.42535625e-01,
        -4.29197538e-01,  7.97724035e-01, -5.68796459e-01,
        -4.10890702e-01, -4.10890702e-01, -1.50755672e-01,
        -7.97724035e-01, -1.50755672e-01, -3.92232270e-01,
         9.43398113e+00, -1.05999788e-01, -1.05999788e-01,
        -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
        -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
        -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
        -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
        -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
        -1.05999788e-01, -1.50755672e-01, -1.05999788e-01,
        -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
        -1.05999788e-01, -1.50755672e-01, -1.05999788e-01,
        -1.50755672e-01, -1.50755672e-01, -1.05999788e-01,
        -1.50755672e-01, -1.50755672e-01, -1.05999788e-01,
        -1.05999788e-01, -1.50755672e-01, -1.50755672e-01,
        -1.50755672e-01, -1.05999788e-01, -1.05999788e-01,
        -1.05999788e-01, -1.50755672e-01, -2.15665546e-01,
        -1.85695338e-01, -2.15665546e-01, -2.67261242e-01,
        -1.05999788e-01, -2.42535625e-01, -1.05999788e-01,
        -2.15665546e-01, -1.85695338e-01, -2.15665546e-01,
        -1.85695338e-01, -1.05999788e-01,  1.87082869e+00,
        -1.87082869e+00,  8.35531692e-01, -8.35531692e-01,
         1.93309133e+00, -1.93309133e+00],
       [-1.67441914e+00, -1.19523159e+00, -6.53912840e-01,
        -1.57589457e+00, -9.73440458e-01, -1.05999788e-01,
        -1.05999788e-01, -6.54653671e-01, -1.05999788e-01,
        -5.51677284e-01,  3.44342023e+00, -1.85695338e-01,
        -3.33333333e-01, -1.05999788e-01, -2.42535625e-01,
        -4.29197538e-01,  7.97724035e-01, -5.68796459e-01,
        -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
        -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
        -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
        -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
        -1.05999788e-01, -1.50755672e-01, -1.05999788e-01,
        -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
        -1.05999788e-01, -1.50755672e-01, -1.05999788e-01,
        -1.50755672e-01, -1.50755672e-01, -1.05999788e-01,
        -1.50755672e-01, -1.50755672e-01, -1.05999788e-01,
        -1.05999788e-01, -1.50755672e-01, -1.50755672e-01,
        -1.50755672e-01, -1.05999788e-01, -1.05999788e-01,
        -1.05999788e-01, -1.50755672e-01, -2.15665546e-01,
        -1.85695338e-01, -2.15665546e-01, -2.67261242e-01,
        -1.05999788e-01, -2.42535625e-01, -1.05999788e-01,
        -2.15665546e-01, -1.85695338e-01, -2.15665546e-01,
        -1.85695338e-01, -1.05999788e-01,  1.87082869e+00,
        -1.87082869e+00,  8.35531692e-01, -8.35531692e-01,
         1.93309133e+00, -1.93309133e+00],
       [-1.63592675e+00, -1.16267307e+00, -6.53912840e-01,
        -1.57589457e+00, -9.73440458e-01, -1.05999788e-01,
        -1.05999788e-01, -6.54653671e-01, -1.05999788e-01,
         1.81265393e+00, -2.90408935e-01, -1.85695338e-01,
        -3.33333333e-01, -1.05999788e-01, -2.42535625e-01,
        -4.29197538e-01,  7.97724035e-01, -5.68796459e-01,
        -4.10890702e-01, -4.10890702e-01, -1.50755672e-01,
        -7.97724035e-01, -1.50755672e-01, -3.92232270e-01,
        -1.05999788e-01, -1.05999788e-01,  9.43398113e+00,
        -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
        -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
        -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
        -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
        -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
        -1.05999788e-01, -1.50755672e-01, -1.05999788e-01,
        -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
        -1.05999788e-01, -1.50755672e-01, -1.05999788e-01,
        -1.50755672e-01, -1.50755672e-01, -1.05999788e-01,
        -1.50755672e-01, -1.50755672e-01, -1.05999788e-01,
        -1.05999788e-01, -1.50755672e-01, -1.50755672e-01,
        -1.50755672e-01, -1.05999788e-01, -1.05999788e-01,
        -1.05999788e-01, -1.50755672e-01, -2.15665546e-01,
        -1.85695338e-01, -2.15665546e-01, -2.67261242e-01,
        -1.05999788e-01, -2.42535625e-01, -1.05999788e-01,
        -2.15665546e-01, -1.85695338e-01, -2.15665546e-01,
        -1.85695338e-01, -1.05999788e-01,  1.87082869e+00,
        -1.87082869e+00,  8.35531692e-01, -8.35531692e-01,
         1.93309133e+00, -1.93309133e+00],
       [-1.59743435e+00, -1.20058661e+00, -6.53912840e-01,
        -1.57589457e+00, -9.73440458e-01, -1.05999788e-01,
        -1.05999788e-01, -6.54653671e-01, -1.05999788e-01,
        -5.51677284e-01, -2.90408935e-01, -1.85695338e-01,
         3.00000000e+00, -1.05999788e-01, -2.42535625e-01,
        -4.29197538e-01, -1.25356634e+00, -5.68796459e-01,
         2.43373723e+00, -4.10890702e-01, -1.50755672e-01,
        -7.97724035e-01, -1.50755672e-01, -3.92232270e-01,
```

```
       -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
       -1.05999788e-01, -1.50755672e-01, -1.05999788e-01,
       -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
       -1.05999788e-01, -1.50755672e-01, -1.05999788e-01,
       -1.50755672e-01, -1.50755672e-01, -1.05999788e-01,
       -1.50755672e-01, -1.50755672e-01, -1.05999788e-01,
       -1.05999788e-01, -1.50755672e-01, -1.50755672e-01,
       -1.50755672e-01, -1.05999788e-01, -1.05999788e-01,
       -1.05999788e-01, -1.50755672e-01, -2.15665546e-01,
       -1.85695338e-01, -2.15665546e-01, -2.67261242e-01,
       -1.05999788e-01, -2.42535625e-01, -1.05999788e-01,
       -2.15665546e-01, -1.85695338e-01, -2.15665546e-01,
       -1.85695338e-01, -1.05999788e-01,  1.87082869e+00,
       -1.87082869e+00,  8.35531692e-01, -8.35531692e-01,
        1.93309133e+00, -1.93309133e+00]])
```

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`.

TASK 3

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`

[10]:
```python
X_train, X_test, Y_train, Y_test = train_test_split( X, Y, test_size=0.2, random_state=2)
print ('Train set:', X_train.shape,  Y_train.shape)
print ('Test set:', X_test.shape,  Y_test.shape)
```

```
Train set: (72, 83) (72,)
Test set: (18, 83) (18,)
```

we can see we only have 18 test samples.

[11]:
```python
Y_test.shape
```

[11]: `(18,)`

TASK 4

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`.

[12]:
```python
parameters ={'C':[0.01,0.1,1],
             'penalty':['l2'],
             'solver':['lbfgs']}

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

[13]:
```python
parameters ={"C":[0.01,0.1,1],'penalty':['l2'], 'solver':['lbfgs']}# l1 lasso l2 ridge
lr=LogisticRegression()
```

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_`.

[14]:
```python
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': 'lbfgs'}
accuracy : 0.8464285714285713
```

TASK 5

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

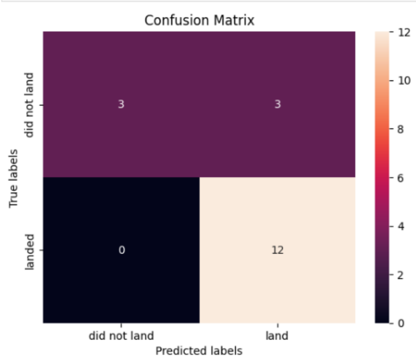
Calculate the accuracy on the test data using the method `score`:

[15]:
```python
logreg_cv.score(X_test, Y_test)
```

[15]: `0.8333333333333334`

Lets look at the confusion matrix:

[16]:
```python
yhat=logreg_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)
```



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 then create a `GridSearchCV` object `svm_cv` with cv = 10. Fit the object to find the best parameters from the dictionary `parameters`.

[17]:
```python
parameters = {'kernel':('linear', 'rbf','poly','rbf', 'sigmoid'),
              'C': np.logspace(-3, 3, 5),
              'gamma':np.logspace(-3, 3, 5)}
svm = SVC()
```

```
[18]: grid_search = GridSearchCV(svm, parameters, cv=10)
      svm_cv = grid_search.fit(X_train, Y_train)
```

```
[19]: 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, 'kernel': 'sigmoid'}
accuracy : 0.8482142857142856
```

TASK 7

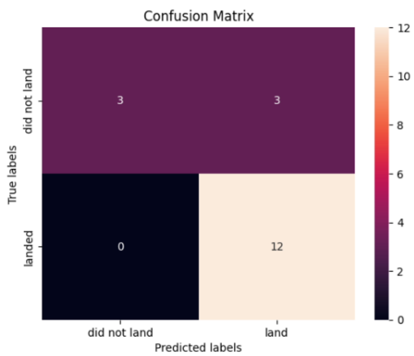
Calculate the accuracy on the test data using the method `score`:

```
[20]: svm_cv.score(X_test, Y_test)
```

```
[20]: 0.8333333333333334
```

We can plot the confusion matrix

```
[21]: 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 cv = 10. Fit the object to find the best parameters from the dictionary `parameters`.

```
[41]: 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()
```

```
[40]: grid_search = GridSearchCV(tree, parameters, cv=10)
      tree_cv = grid_search.fit(X_train, Y_train)
```

```
tuned hyperparameters :(best parameters)  {'criterion': 'gini', 'max_depth': 6, 'max_features': 'sqrt', 'min_samples_leaf': 2, 'min_samples_split': 5, 'splitter': 'random'}
accuracy : 0.8732142857142856
```

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

```
tuned hpyerparameters :(best parameters)  {'criterion': 'gini', 'max_depth': 12, 'max_features': 'sqrt', 'min_samples_leaf': 1, 'min_samples_split': 2, 'splitter': 'best'}
accuracy : 0.8732142857142857
```

TASK 9

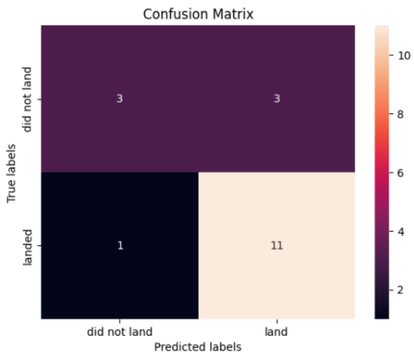
Calculate the accuracy of tree_cv on the test data using the method `score`:

```
[31]: tree_cv.score(X_test, Y_test)
```

```
[31]: 0.7777777777777778
```

We can plot the confusion matrix

```
[32]: yhat = tree_cv.predict(X_test)
      plot_confusion_matrix(Y_test,yhat)
```



TASK 10

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`.

[33]:
```
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)

[45]:
```
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 : 0.8482142857142858
```

TASK 11

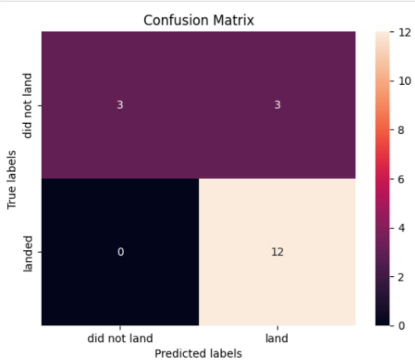
Calculate the accuracy of knn_cv on the test data using the method `score`:

[36]:
```
knn_cv.score(X_test, Y_test)
```

[36]:
```
0.8333333333333334
```

We can plot the confusion matrix

[37]:
```
yhat = knn_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)
```



TASK 12

Find the method performs best:

[38]:
```
print('Accuracy for Logistics Regression method:', logreg_cv.score(X_test, Y_test))
print( 'Accuracy for Support Vector Machine method:', svm_cv.score(X_test, Y_test))
print('Accuracy for Decision tree method:', tree_cv.score(X_test, Y_test))
print('Accuracy for K nearsdt neighbors method:', knn_cv.score(X_test, Y_test))
```

```
Accuracy for Logistics Regression method: 0.8333333333333334
Accuracy for Support Vector Machine method: 0.8333333333333334
Accuracy for Decision tree method: 0.7777777777777778
Accuracy for K nearsdt neighbors method: 0.8333333333333334
```

Authors

Pratiksha Verma

Change Log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2022-11-09	1.0	Pratiksha Verma	Converted initial version to Jupyterlite