Import Libraries & Read in Data

```
import pandas as pd
import numpy as np
import plotly.express as px
from sklearn.model_selection import train_test_split
import plotly.graph_objects as go
import seaborn as sns

# import google drive
from google.colab import drive
drive.mount('/content/drive/')

# Change directory to google drive- Just upload the file right into the drive you want(Uchenn
%cd /content/drive/My Drive/

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

#define titanic - you'd need this going forward
nasa = pd.read_csv('nasa_data.csv')
nasa.head()
```

```
Mounted at /content/drive/
/content/drive/My Drive
```

	unit_number	time_in_cycles	Altitud	Mach Number	TRA	T2	T24	T30	T50
			-0.0007	-0.0004	100.0	518.67	641.82	1589.70	1400.60
			0.0019	-0.0003	100.0	518.67	642.15	1591.82	1403.14
2	1	3	-0.0043	0.0003	100.0	518.67	642.35	1587.99	1404.20
3	1	4	0.0007	0.0000	100.0	518.67	642.35	1582.79	1401.87
4	1	5	-0.0019	-0.0002	100.0	518.67	642.37	1582.85	1406.22

Saved successfully!

```
#defining a new target variable based on a minimum threshold of 25
target = 25
label_positive =nasa['target'] <= target
nasa['label_target']=1
nasa.loc[label_positive,'label_target'] = 0

#Unit number not likely to be relevant to the process, also condition is just the data set #
nasa.drop(columns=['max_cycles','target','unit_number','condition'],inplace = True)
```

Split into train and test set

Note: Final Test set not included so technically , test set referred to here is the validation set

```
X = nasa.drop(['label_target'], axis=1)
y = nasa['label_target']

#splitting the data set (note we already have an actual test set, so this test set here is th
X_train, X_val, y_train, y_val = train_test_split( X, y, test_size=0.33, random_state=42,stra


#confirming that the split was done (67% to 33%)
for dataset in [y_train, y_val]:
    print(round(len(dataset) / len(y), 2))
```

```
0.67
0.33
```

```
#Display X_train
X_train.head()
```

	time_in_cycles	Altitud	Mach Number	TRA	T2	T24	T30	T50	P2	P
159047	299	35.0020	0.8417	100.0	449.44	555.10	1369.15	1140.66	5.48	7
32086	64	42.0064	0.8400	100.0	445.00	549.32	1352.52	1114.88	3.91	5
155840	196	35.0050	0.8400	100.0	449.44	555.15	1363.23	1126.76	5.48	7
134494	98	20.0057	0.7007	100.0	491.19	607.12	1479.43	1245.62	9.35	13
		6	0.6215	60.0	462.54	537.00	1261.28	1051.99	7.05	9

Saved successfully!

Write out all data

```
X_train.to_csv('nasatrain_features.csv', index=False)
X_val.to_csv('nasaval_features.csv', index=False)


y_train.to_csv('nasatrain_labels.csv', index=False)
y_val.to_csv('nasaval_labels.csv', index=False)



#Read in Training Data
tr_features = pd.read_csv('nasatrain_features.csv')
tr_labels = pd.read_csv('nasatrain_labels.csv')
```

```
#Define Results to print
def print_results(results):
    print('BEST PARAMS: {}\n'.format(results.best_params_))

    means = results.cv_results_['mean_test_score']
    stds = results.cv_results_['std_test_score']
    for mean, std, params in zip(means, stds, results.cv_results_['params']):
        print('{} (+/-{}) for {}'.format(round(mean, 3), round(std * 2, 3), params))
```

kNN

```
#importing Libraries
from sklearn.neighbors import KNeighborsClassifier
import joblib
import pandas as pd
from sklearn.model_selection import GridSearchCV
import warnings
warnings.filterwarnings('ignore', category=FutureWarning)
warnings.filterwarnings('ignore', category=DeprecationWarning)


# To show list of hyperparameters that we can tune
KNeighborsClassifier()
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
```

Saved successfully!

```
#Fitting the Model and Evaluating
knn = KNeighborsClassifier()
parameters = {
    'n_neighbors': [1,3, 5,11,15,20],
    # to test 1,3,5,10,15,20 leaves,
    'weights':['uniform','distance']

}
#using GridSearchCV to loop through predefined hyperparameters and fit your estimator (model)
cv = GridSearchCV(knn,parameters, cv = 5)
#cv = 5 meaning it will run 5-fold validation for each hyperparameter combination
cv.fit(tr_features,tr_labels.values.ravel())
# we use ravel for the labels to convert it to an array, since the label is usually just one
print_results(cv)
```

```
BEST PARAMS: {'n_neighbors': 15, 'weights': 'distance'}

0.943 (+/-0.002) for {'n_neighbors': 1, 'weights': 'uniform'}
0.943 (+/-0.002) for {'n_neighbors': 1, 'weights': 'distance'}
0.951 (+/-0.002) for {'n_neighbors': 3, 'weights': 'uniform'}
```

```
0.951 (+/-0.002) for {'n_neighbors': 3, 'weights': 'distance'}
0.953 (+/-0.001) for {'n_neighbors': 5, 'weights': 'uniform'}
0.953 (+/-0.001) for {'n_neighbors': 5, 'weights': 'distance'}
0.954 (+/-0.001) for {'n_neighbors': 11, 'weights': 'uniform'}
0.955 (+/-0.001) for {'n_neighbors': 11, 'weights': 'distance'}
0.954 (+/-0.001) for {'n_neighbors': 15, 'weights': 'uniform'}
0.955 (+/-0.002) for {'n_neighbors': 15, 'weights': 'distance'}
0.954 (+/-0.002) for {'n_neighbors': 20, 'weights': 'uniform'}
0.955 (+/-0.002) for {'n_neighbors': 20, 'weights': 'distance'}
```

```
cv.best_estimator_
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=15, p=2,
                     weights='distance')
```

Write out pickled model

We pickle the model by saving it and writing it to a file that can be used to compare with other hyperparameters performance

```
joblib.dump(cv.best_estimator_, '../../../NASAkNN_model.pkl')
```

```
['../../../NASAkNN_model.pkl']
```

```
cv = cv.best_estimator_
#Predicting the labels using the optimized hyperparameters
tr_labelspredict = cv.predict(tr_features)
```

Saved successfully!

```
# To view accuracy score, recall score, precision score and f1 score
from sklearn.metrics import accuracy_score, recall_score, precision_score, f1_score

 #Train Scores

acc_train = accuracy_score(y_train,tr_labelspredict)
p_score_train = precision_score(y_train,tr_labelspredict)
r_score_train = recall_score(y_train,tr_labelspredict)
f1_score_train = f1_score(y_train,tr_labelspredict)

 #Test Scores

val_labelspredict = cv.predict(X_val)
acc_test = accuracy_score(y_val,val_labelspredict)
p_score_test = precision_score(y_val,val_labelspredict)
r_score_test = recall_score(y_val,val_labelspredict)
f1_score_test = f1_score(y_val,val_labelspredict)

print(f'The Accuracy score for the training set is:{acc_train}')
```

```
print(f'The Precision score for the training set is:{p_score_train}')
print(f'The Recall score for the training set is:{r_score_train}')
print(f'The F1 score for the training set is:{f1_score_train}')
print('------------------------------------------------------------')
print(f'The Accuracy score for the validation set is:{acc_test}')
print(f'The Precision score for the validation set is:{p_score_test}')
print(f'The Recall score for the validation set is:{r_score_test}')
print(f'The F1 score for the validation set is:{f1_score_test}')
# train_results = [acc_train,p_score_train,r_score_train,f1_score_train]
# test_results = [acc_test,p_score_test,r_score_test,f1_score_test]
```

```
The Accuracy score for the training set is:1.0
The Precision score for the training set is:1.0
The Recall score for the training set is:1.0
The F1 score for the training set is:1.0
------------------------------------------------------------
The Accuracy score for the validation set is:0.9575388801753624
The Precision score for the validation set is:0.9691109942135718
The Recall score for the validation set is:0.9833674950892476
The F1 score for the validation set is:0.9761871959813907
```