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()
```

```
Drive already mounted at /content/drive/; to attempt to forcibly remount, call drive.mou
/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


```
#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	F
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
[illegible]	[illegible]	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')

val_features = pd.read_csv('nasaval_features.csv')
```

```
val_labels = pd.read_csv('nasaval_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))
```

Building the Regression QuAM

Let's load some regressor from scikit-learn and apply them. We specifically apply Lasso, k -NN regressor and DT Regressors. We use the R2 score for validation. Remember R2 is a score and the best score is the bigest score

Linear Regression Using Lasso - L1 Regularizer

```
from sklearn.linear_model import Lasso
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
```

Saved successfully!

```
lasso_regressor.fit(X_train, y_train)
```

```
Lasso(alpha=1.0, copy_X=True, fit_intercept=True, max_iter=1000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
```

Prediction & Evaluation

```
yhat_val = lasso_regressor.predict(X_val)

display(r2_score(y_val, yhat_val))
display(mean_squared_error(y_val, yhat_val))
display(mean_absolute_error(y_val, yhat_val))
```

```
0.15080036916081685
```

Linear Regression Using Ridge - L2 Regularizer

```
#Import Libraries
from sklearn.linear_model import Ridge
```

```
ridge_regressor = Ridge(alpha=Lambda)
ridge_regressor.fit(X_train, y_train)
```

```
Ridge(alpha=0.0001, copy_X=True, fit_intercept=True, max_iter=None,
      normalize=False, random_state=None, solver='auto', tol=0.001)
```

Prediction & Evaluation

```
yhat_val = ridge_regressor.predict(X_val)

display(r2_score(y_val, yhat_val))
display(mean_squared_error(y_val, yhat_val))
display(mean_absolute_error(y_val, yhat_val))
```

```
0.4888399870176051
0.052003338904498306
[illegible]
```

Saved successfully!

Using kNN Regressor

```
#import libraries
from  sklearn.neighbors import KNeighborsRegressor
```

```
knn_regressor = KNeighborsRegressor()
knn_regressor.fit(X_train, y_train)
```

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

Prediction & Evaluation

```
yhat_val = knn_regressor.predict(X_val)

display(r2_score(y_val, yhat_val))
display(mean_squared_error(y_val, yhat_val))
display(mean_absolute_error(y_val, yhat_val))
```

```
0.6738118316795867
0.03318505640696159
0.058946692114363465
```

Saved successfully!