Scale Features and Build Model

Scales Raw Features

Import CSV of Aggregated Darshan Logs
Apply Log10 and Percent Scaling

In [1]:
```
import os
import pandas as pd
import numpy as np
import math
import matplotlib.pyplot as plt
```

In [2]:
```
df = pd.read_csv("./raws.csv",lineterminator='\n', error_bad_lines=False )
#df.mean()
```

In [3]:
```
df = df.drop(df.columns[0],axis = 1)
df = df.drop(df.columns[0],axis = 1)
f = pd.DataFrame()
```

In [4]:
```
df
```

Out[4]:

	posix_read_time	posix_write_time	posix_meta_time	posix_bytes_read	posix_bytes_read_100
0	0.000000	0.000000	0.000000	0.000000e+00	0.0
1	104.611641	10.024055	20.060841	2.390891e+10	147688.0
2	124.560730	42.051125	54.839272	5.019637e+10	332059.0
3	25763.292969	582.297363	24.895737	5.488943e+12	30785.0
4	154.534821	681.548279	658.484985	2.293203e+10	588029.0
...	...	...	...	...	...
875282	138.354477	82.278084	194.485565	5.593977e+10	216146.0
875283	54.443073	231.440857	25.271391	1.465277e+09	3099.0
875284	0.000000	0.000000	0.000000	0.000000e+00	0.0
875285	0.000000	0.000000	0.000000	0.000000e+00	0.0
875286	227.063828	191.747269	172.671997	1.077412e+11	775359.0

875287 rows × 50 columns

In [5]:
```
df = df.dropna(axis=0, how='any')
df
```

Out[5]:

	posix_read_time	posix_write_time	posix_meta_time	posix_bytes_read	posix_bytes_read_100
0	0.000000	0.000000	0.000000	0.000000e+00	0.0
1	104.611641	10.024055	20.060841	2.390891e+10	147688.0
2	124.560730	42.051125	54.839272	5.019637e+10	332059.0
3	25763.292969	582.297363	24.895737	5.488943e+12	30785.0
4	154.534821	681.548279	658.484985	2.293203e+10	588029.0
...	...	...	...	...	...
875282	138.354477	82.278084	194.485565	5.593977e+10	216146.0
875283	54.443073	231.440857	25.271391	1.465277e+09	3099.0
875284	0.000000	0.000000	0.000000	0.000000e+00	0.0
875285	0.000000	0.000000	0.000000	0.000000e+00	0.0
875286	227.063828	191.747269	172.671997	1.077412e+11	775359.0

875287 rows × 50 columns

In [6]:
```
#files
f['log10_p_files'] = np.log10(df['posix_number_of_files'])
f['log10_l_files'] = np.log10(df['lustre_number_of_files'])
```

```
/Users/dave/opt/anaconda3/lib/python3.8/site-packages/pandas/core/arrayli
ke.py:358: RuntimeWarning: divide by zero encountered in log10
  result = getattr(ufunc, method)(*inputs, **kwargs)
```

In [7]:
```
#accesses
df['p_accesses'] = np.log10(df['posix_reads'] + df['posix_writes'])
f['log10_p_accesses'] = np.log10(df['p_accesses'])

f['log10_p_accesses']
```

```
/Users/dave/opt/anaconda3/lib/python3.8/site-packages/pandas/core/arrayli
ke.py:358: RuntimeWarning: invalid value encountered in log10
  result = getattr(ufunc, method)(*inputs, **kwargs)
```

Out[7]:
```
0              NaN
1         0.774119
2         0.804583
3         0.841957
4         0.838513
             ...
875282    0.802713
875283    0.723940
875284         NaN
875285         NaN
875286    0.831411
Name: log10_p_accesses, Length: 875287, dtype: float64
```

In [8]:
```
#bytes
f['log10_p_bytes'] = np.log10(df['posix_bytes_read'] )
```

In [9]:
```
f['log10_p_opens'] = np.log10(df['posix_opens'])
f['log10_p_seeks'] = np.log10(df['posix_seeks'])
f['log10_p_stats'] = np.log10(df['posix_stats'])
f['log10_p_mode'] = np.log10(df['posix_renamed_mode'])
```

In [10]:
```
f['log10_l_n_osts'] = np.log10(df['lustre_number_of_osts'])
f['log10_l_stripe_w'] = np.log10(df['lustre_stripe_width'])
f['log10_l_mdts'] = np.log10(df['lustre_mdts'])
```

In [11]:
```
f['log10_p_nprocs'] = np.log10(df['nprocs'])
f['log10_p_falign'] = np.log10(df['posix_f_align'])
f['log10_p_malign'] = np.log10(df['posix_m_align'])
```

In [12]:
```
f['perc_p_reads'] = df['posix_reads'] /df['p_accesses']
f['perc_p_writes'] = df['posix_writes'] /df['p_accesses']
```

In [14]:
```
df['histogram_r_sum'] = df['posix_bytes_read_100'] + df['posix_bytes_re
df['posix_bytes_read_4M'] + df['posix_bytes_read_10M'] + df['posix_byte
df['posix_bytes_read_1G'] + df['posix_bytes_read_PLUS'] + df['posix_byt

f['perc_p_bytes_read_100'] = df['posix_bytes_read_100']/df['histogram_r
f['perc_p_bytes_read_1K'] = df['posix_bytes_read_1K']/df['histogram_r_s
f['perc_p_bytes_read_10K'] = df['posix_bytes_read_10K']/df['histogram_r
f['perc_p_bytes_read_100K'] = df['posix_bytes_read_100K']/df['histogram
f['perc_p_bytes_read_1M'] = df['posix_bytes_read_1lM']/df['histogram_r_
f['perc_p_bytes_read_4M'] = df['posix_bytes_read_4M']/df['histogram_r_s
f['perc_p_bytes_read_10M'] = df['posix_bytes_read_10M']/df['histogram_r
f['perc_p_bytes_read_100M'] = df['posix_bytes_read_100M']/df['histogram
f['perc_p_bytes_read_1G'] = df['posix_bytes_read_1G']/df['histogram_r_s
f['perc_p_bytes_read_PLUS'] = df['posix_bytes_read_PLUS']/df['histogram
```

In [15]:
```
df['histogram_w_sum'] = df['posix_bytes_write_100'] + df['posix_bytes_write
df['posix_bytes_write_10K'] + df['posix_bytes_write_100K'] + df['posix_byte
df['posix_bytes_write_4M'] + df['posix_bytes_write_10M'] + df['posix_bytes_
df['posix_bytes_write_1G'] + df['posix_bytes_write_PLUS']

f['perc_p_bytes_write_100'] = df['posix_bytes_write_100']/df['histogram_w_s
f['perc_p_bytes_write_1K'] = df['posix_bytes_write_1K']/df['histogram_w_sum
f['perc_p_bytes_write_10K'] = df['posix_bytes_write_10K']/df['histogram_w_s
f['perc_p_bytes_write_100K'] = df['posix_bytes_write_100K']/df['histogram_w
f['perc_p_bytes_write_1M'] = df['posix_bytes_write_1M']/df['histogram_w_sum
f['perc_p_bytes_write_4M'] = df['posix_bytes_write_4M']/df['histogram_w_sum
f['perc_p_bytes_write_10M'] = df['posix_bytes_write_10M']/df['histogram_w_s
f['perc_p_bytes_write_100M'] = df['posix_bytes_write_100M']/df['histogram_w
f['perc_p_bytes_write_1G'] = df['posix_bytes_write_1G']/df['histogram_w_sum
f['perc_p_bytes_write_PLUS'] = df['posix_bytes_write_PLUS']/df['histogram_w


f = f.replace(-np.inf, -1)
f = f.replace(np.nan, 0)
```

In [16]:
```
df['time'] = df['posix_write_time'] + df['posix_read_time'] + df['posix_met
df['bytes'] = df['posix_bytes_read'] + df['posix_bytes_write']

t = pd.DataFrame()
f['throughput'] = df['bytes'] / df['time']
f = f[f['throughput'] >0]
```

In [17]:
```
#delete columns with all zeros
f = f.loc[:, (f != 0).any(axis=0)]

#remove infinite values
f = f.replace([np.inf, -np.inf], np.nan).dropna(axis=0)

f.max()
```

Out[17]:
```
log10_p_files              5.086100e+00
log10_l_files              5.086100e+00
log10_p_accesses           1.015048e+00
log10_p_bytes              1.448265e+01
log10_p_opens              8.742811e+00
log10_p_seeks              1.015993e+01
log10_p_stats              7.814442e+00
log10_p_mode               7.727321e+00
log10_l_n_osts             2.556303e+00
log10_l_stripe_w           6.871490e+00
log10_l_mdts               0.000000e+00
log10_p_nprocs             5.546543e+00
log10_p_falign             1.115306e+01
log10_p_malign             6.035555e+00
perc_p_reads               2.161636e+09
perc_p_writes              1.287974e+09
perc_p_bytes_read_100      1.000000e+00
perc_p_bytes_read_1K       1.000000e+00
perc_p_bytes_read_10K      9.331507e+07
perc_p_bytes_read_100K     1.000000e+00
perc_p_bytes_read_1M       1.000000e+00
perc_p_bytes_read_4M       1.000000e+00
perc_p_bytes_read_10M      9.978617e-01
perc_p_bytes_read_100M     1.000000e+00
perc_p_bytes_read_1G       1.000000e+00
perc_p_bytes_write_100     1.000000e+00
perc_p_bytes_write_1K      1.000000e+00
perc_p_bytes_write_10K     1.000000e+00
perc_p_bytes_write_100K    1.000000e+00
perc_p_bytes_write_1M      1.000000e+00
perc_p_bytes_write_4M      1.000000e+00
perc_p_bytes_write_10M     1.000000e+00
perc_p_bytes_write_100M    1.000000e+00
perc_p_bytes_write_1G      1.000000e+00
throughput                 2.344536e+09
dtype: float64
```

```
t['throughput'] = f['throughput']
f = f.drop(labels = 'throughput', axis = 1)
f
```

In [19]:
```
t.max()
```

Out[19]:
```
throughput    2.344536e+09
dtype: float64
```

In [20]:
```
print(t.min())
print(t.max())
```

```
throughput    0.39201
dtype: float64
throughput    2.344536e+09
dtype: float64
```

In [21]:
```
from sklearn.model_selection import train_test_split
import scipy as sp
import random
from sklearn.metrics import r2_score

rseed = 123
t_size = 0.2

train_data, test_data, train_labels, test_labels = train_test_split(f,t, te
```

In [22]:
```
import xgboost as xg
```

In [23]:
```
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_log_error
from sklearn.metrics import mean_absolute_percentage_error
```

In [24]:
```
from sklearn.linear_model import LinearRegression
for i in range(3):

    rseed = random.randint(1,10000)
    print(rseed)
    train_data, test_data, train_labels, test_labels = train_test_split(f,t

    reg = LinearRegression().fit(train_data, train_labels)

    predicted_labels = reg.predict(test_data)

    print("Mean True Value: \t",int(test_labels.mean() ))
    print("Mean Absolute Error: \t", int(mean_absolute_error(test_labels, p
    print("Mean Squared Error: ", mean_squared_error(test_labels, predicted
    print("Root Mean Squared Error: ", mean_squared_error(test_labels, pred
    print("MAPE :" + str(mean_absolute_percentage_error( test_labels, predi
    print("R2: " + str(r2_score(test_labels,predicted_labels)) + "\n")
```

```
5256
Mean True Value:         84655611
Mean Absolute Error:     35306336
Mean Squared Error:  3998930934775507.5
Root Mean Squared Error:  63237100.935886584
MAPE :430.5851114848273
R2: 0.6645808586938563

9498
Mean True Value:         84440338
Mean Absolute Error:     35161045
Mean Squared Error:  3950854968803895.5
Root Mean Squared Error:  62855826.84846247
MAPE :4020.4763055816597
R2: 0.664068592196188

2193
Mean True Value:         83728301
Mean Absolute Error:     35009538
Mean Squared Error:  3930630026111212.5
Root Mean Squared Error:  62694736.829427816
MAPE :366.7762125668867
R2: 0.6631738087493297
```

In [25]:
```
#EXTREME GRADIENT BOOST

for i in range(3):
    rseed = random.randint(1,10000)
    print(rseed)

    train_data, test_data, train_labels, test_labels = train_test_split(f,t
    xgb_r = xg.XGBRegressor(n_estimators = 1000, seed = 123)
    xgb_r.fit(train_data, train_labels)
    predicted_labels = xgb_r.predict(test_data)

    print("Mean True Value: \t",int(test_labels.mean() ))
    print("Mean Absolute Error: \t", int(mean_absolute_error(test_labels, p
    print("Mean Squared Error: ", mean_squared_error(test_labels, predicted
    print("Root Mean Squared Error: ", mean_squared_error(test_labels, pred
    print("MAPE :" + str(mean_absolute_percentage_error( test_labels, predi
    print("R2: " + str(r2_score(test_labels,predicted_labels)) + "\n")
```

```
3363
Mean True Value:         84417914
Mean Absolute Error:     16232655
Mean Squared Error:  1159032349626505.8
Root Mean Squared Error:  34044564.17148714
MAPE :48.98696827810851
R2: 0.9017313477293982

8266
Mean True Value:         84104805
Mean Absolute Error:     16300453
Mean Squared Error:  1176870360451501.0
Root Mean Squared Error:  34305544.16492327
MAPE :19.384029900993173
R2: 0.9001362752350884

8582
Mean True Value:         84536628
Mean Absolute Error:     16329916
Mean Squared Error:  1177729119299233.5
Root Mean Squared Error:  34318058.20991673
MAPE :679.6267639834404
R2: 0.9011427953693474
```

In [26]:
```
from sklearn.tree import DecisionTreeRegressor

for i in range(3):
    rseed = random.randint(1,10000)
    print(rseed)

    train_data, test_data, train_labels, test_labels = train_test_split(f,t

    reg = DecisionTreeRegressor(max_depth = 4)


    reg.fit(train_data, train_labels)
    predicted_labels = reg.predict(test_data)


    print("Mean True Value: \t",int(test_labels.mean() ))
    print("Mean Absolute Error: \t", int(mean_absolute_error(test_labels, p
    print("Mean Squared Error: ", mean_squared_error(test_labels, predicted
    print("Root Mean Squared Error: ", mean_squared_error(test_labels, pred
    print("MAPE :" + str(mean_absolute_percentage_error( test_labels, predi
    print("R2: " + str(r2_score(test_labels,predicted_labels)) + "\n")
```

```
1479
Mean True Value:         84871582
Mean Absolute Error:     28228907
Mean Squared Error:  2842564157355832.0
Root Mean Squared Error:  53315702.72776897
MAPE :1648.03877149611
R2: 0.764575284292099

3502
Mean True Value:         84302211
Mean Absolute Error:     28107752
Mean Squared Error:  2698953271249399.5
Root Mean Squared Error:  51951451.098592035
MAPE :1140.1004613705709
R2: 0.7726844672772891

5677
Mean True Value:         84134723
Mean Absolute Error:     28190851
Mean Squared Error:  2769122901493667.0
Root Mean Squared Error:  52622456.2472493
MAPE :993.7645129368356
R2: 0.7656705470577084
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: