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
import random

from sklearn.model_selection import train_test_split
```

In [2]:
```
df = pd.read_csv("./raws.csv",lineterminator='\n',sep = ',' ,error_bad_line
#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.columns
```

Out[4]:
```
Index(['posix_read_time', 'posix_write_time', 'posix_meta_time',
       'posix_bytes_read', 'posix_bytes_read_100', 'posix_bytes_read_1K',
       'posix_bytes_read_10K', 'posix_bytes_read_100K', 'posix_bytes_read
_11M',
       'posix_bytes_read_4M', 'posix_bytes_read_10M', 'posix_bytes_read_1
00M',
       'posix_bytes_read_1G', 'posix_bytes_read_PLUS', 'posix_bytes_writ
e',
       'posix_bytes_write_100', 'posix_bytes_write_1K',
       'posix_bytes_write_10K', 'posix_bytes_write_100K',
       'posix_bytes_write_1M', 'posix_bytes_write_4M', 'posix_bytes_write
_10M',
       'posix_bytes_write_100M', 'posix_bytes_write_1G',
       'posix_bytes_write_PLUS', 'posix_opens', 'posix_reads', 'posix_wri
tes',
       'posix_seeks', 'posix_stats', 'posix_mmaps', 'posix_fsyncs',
       'posix_fdsyncs', 'posix_rename_sources', 'posix_rename_targets',
       'posix_renamed_from', 'posix_renamed_mode', 'posix_number_of_file
s',
       'nprocs', 'posix_f_align', 'posix_m_align', 'lustre_number_of_file
s',
       'lustre_mdts', 'lustre_osts', 'lustre_stripe_size',
       'lustre_stripe_offset', 'lustre_stripe_width', 'lustre_number_of_o
sts',
       'jobid', 'path'],
      dtype='object')
```

In [5]:

```

df = df.dropna(axis=0, how='any')
df.columns
```

Out[5]:

```
Index(['posix_read_time', 'posix_write_time', 'posix_meta_time',
       'posix_bytes_read', 'posix_bytes_read_100', 'posix_bytes_read_1K',
       'posix_bytes_read_10K', 'posix_bytes_read_100K', 'posix_bytes_read
_1lM',
       'posix_bytes_read_4M', 'posix_bytes_read_10M', 'posix_bytes_read_1
00M',
       'posix_bytes_read_1G', 'posix_bytes_read_PLUS', 'posix_bytes_writ
e',
       'posix_bytes_write_100', 'posix_bytes_write_1K',
       'posix_bytes_write_10K', 'posix_bytes_write_100K',
       'posix_bytes_write_1M', 'posix_bytes_write_4M', 'posix_bytes_write
_10M',
       'posix_bytes_write_100M', 'posix_bytes_write_1G',
       'posix_bytes_write_PLUS', 'posix_opens', 'posix_reads', 'posix_wri
tes',
       'posix_seeks', 'posix_stats', 'posix_mmaps', 'posix_fsyncs',
       'posix_fdsyncs', 'posix_rename_sources', 'posix_rename_targets',
       'posix_renamed_from', 'posix_renamed_mode', 'posix_number_of_file
s',
       'nprocs', 'posix_f_align', 'posix_m_align', 'lustre_number_of_file
s',
       'lustre_mdts', 'lustre_osts', 'lustre_stripe_size',
       'lustre_stripe_offset', 'lustre_stripe_width', 'lustre_number_of_o
sts',
       'jobid', 'path'],
      dtype='object')
```

In [6]:

```
#files
f['log10_p_files'] = df['posix_number_of_files']  + 1
f['log10_l_files'] = df['lustre_number_of_files']
```

In [7]:

```
#accesses
df['p_accesses'] = df['posix_reads'] + df['posix_writes']
f['log10_p_accesses'] = df['p_accesses']

f['log10_p_accesses']
```

Out[7]:

```
0                0.0
1           880136.0
2          2379598.0
3          8903411.0
4          7846387.0
             ...
875282     2234152.0
875283      197651.0
875284           0.0
875285           0.0
875286     6065006.0
Name: log10_p_accesses, Length: 875287, dtype: float64
```

In [8]:
```
#bytes
f['log10_p_bytes'] = df['posix_bytes_read']
```

In [9]:
```
f['log10_p_opens'] = df['posix_opens']
f['log10_p_seeks'] = df['posix_seeks']
f['log10_p_stats'] = df['posix_stats']
f['log10_p_mode'] = df['posix_renamed_mode']
```

In [10]:
```
f['log10_l_n_osts'] = df['lustre_number_of_osts']
f['log10_l_stripe_w'] = df['lustre_stripe_width']
f['log10_l_mdts'] = df['lustre_mdts']
```

In [11]:
```
f['log10_p_nprocs'] = df['nprocs']
f['log10_p_falign'] = df['posix_f_align']
f['log10_p_malign'] = df['posix_m_align']
```

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

In [13]:
```
f['perc_p_bytes_read_100'] = df['posix_bytes_read_100']
f['perc_p_bytes_read_1K'] = df['posix_bytes_read_1K']
f['perc_p_bytes_read_10K'] = df['posix_bytes_read_10K']
f['perc_p_bytes_read_100K'] = df['posix_bytes_read_100K']
f['perc_p_bytes_read_1M'] = df['posix_bytes_read_1lM']
f['perc_p_bytes_read_4M'] = df['posix_bytes_read_4M']
f['perc_p_bytes_read_10M'] = df['posix_bytes_read_10M']
f['perc_p_bytes_read_100M'] = df['posix_bytes_read_100M']
f['perc_p_bytes_read_1G'] = df['posix_bytes_read_1G']
f['perc_p_bytes_read_PLUS'] = df['posix_bytes_read_PLUS']
```

In [14]:
```
f['perc_p_bytes_write_100'] = df['posix_bytes_write_100']
f['perc_p_bytes_write_1K'] = df['posix_bytes_write_1K']
f['perc_p_bytes_write_10K'] = df['posix_bytes_write_10K']
f['perc_p_bytes_write_100K'] = df['posix_bytes_write_100K']
f['perc_p_bytes_write_1M'] = df['posix_bytes_write_1M']
f['perc_p_bytes_write_4M'] = df['posix_bytes_write_4M']
f['perc_p_bytes_write_10M'] = df['posix_bytes_write_10M']
f['perc_p_bytes_write_100M'] = df['posix_bytes_write_100M']
f['perc_p_bytes_write_1G'] = df['posix_bytes_write_1G']
f['perc_p_bytes_write_PLUS'] = df['posix_bytes_write_PLUS']


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

In [15]:
```
df['time'] = df['posix_write_time'].astype('float') + df['posix_read_time']
```

In [16]:
```
df['bytes'] = df['posix_bytes_read'].astype('float') + df['posix_bytes_writ
```

In [17]:
```
#df = df[df['bytes'] >99999999]
```

In [18]:

```
f['throughput'] = df['bytes'].astype('float') / df['time']

f = f[f['throughput'] >0]
```

In [19]:

```
#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[19]:

```
log10_p_files              1.219280e+05
log10_l_files              1.219270e+05
log10_p_accesses           2.251942e+10
log10_p_bytes              3.038456e+14
log10_p_opens              5.531094e+08
log10_p_seeks              1.445220e+10
log10_p_stats              6.522921e+07
log10_p_mode               5.337293e+07
log10_l_n_osts             3.600000e+02
log10_l_stripe_w           7.438575e+06
log10_l_mdts               1.000000e+00
log10_p_nprocs             3.520000e+05
log10_p_falign             1.422540e+11
log10_p_malign             1.085312e+06
perc_p_reads               2.237846e+10
perc_p_writes              1.302770e+10
perc_p_bytes_read_100      5.221517e+08
perc_p_bytes_read_1K       2.074657e+10
perc_p_bytes_read_10K      1.536278e+09
perc_p_bytes_read_100K     1.515506e+08
perc_p_bytes_read_1M       4.044503e+08
perc_p_bytes_read_4M       6.561462e+07
perc_p_bytes_read_10M      2.083200e+06
perc_p_bytes_read_100M     2.872090e+05
perc_p_bytes_read_1G       1.792000e+06
perc_p_bytes_write_100     1.302770e+10
perc_p_bytes_write_1K      2.852127e+09
perc_p_bytes_write_10K     3.867477e+08
perc_p_bytes_write_100K    8.347452e+07
perc_p_bytes_write_1M      1.357245e+07
perc_p_bytes_write_4M      3.839488e+06
perc_p_bytes_write_10M     6.190660e+05
perc_p_bytes_write_100M    1.249280e+06
perc_p_bytes_write_1G      1.937500e+04
throughput                 2.344536e+09
dtype: float64
```

In [20]:

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

Out[20]:

	log10_p_files	log10_l_files	log10_p_accesses	log10_p_bytes	log10_p_opens	log10_p_seeks
1	800.0	176.0	880136.0	2.390891e+10	8858.0	319241.0
2	361.0	224.0	2379598.0	5.019637e+10	62398.0	1107764.0
3	291.0	290.0	8903411.0	5.488943e+12	8711.0	2010273.0
4	320.0	201.0	7846387.0	2.293203e+10	23158.0	6015926.0
6	429.0	190.0	6647935.0	5.209185e+10	69261.0	4608438.0
...	...	...	...	...	...	...
875280	2.0	1.0	57344.0	3.006477e+10	1808.0	59128.0
875281	14.0	4.0	102439.0	7.467916e+08	193.0	91.0
875282	625.0	124.0	2234152.0	5.593977e+10	35112.0	1035457.0
875283	1089.0	1088.0	197651.0	1.465277e+09	2112.0	12509.0
875286	583.0	128.0	6065006.0	1.077412e+11	119432.0	2750807.0

671063 rows × 34 columns

In [21]:

```
df = df[df.index.isin(t.index)]
t = t.reset_index()
f = f.reset_index()
f = f.drop(f.columns[0] , axis =1)
t = t.drop(t.columns[0] , axis =1)
```

In [22]:

```

from sklearn.preprocessing import StandardScaler
f = StandardScaler().fit_transform(f)
```

In [23]:
```
t
```

Out[23]:

	throughput
0	1.803194e+08
1	2.282342e+08
2	2.083669e+08
3	1.724841e+07
4	4.581690e+07
...	...
671058	5.482767e+08
671059	2.136637e+07
671060	1.362498e+08
671061	1.942413e+07
671062	1.842942e+08

671063 rows × 1 columns

In [24]:
```
print(t.min())
print(t.max())
```

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

In [25]:
```
rseed = 0
t_size = 0.1
```

In [26]:
```
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 [27]:

```
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")
```

```
466
Mean True Value:         84543501
Mean Absolute Error:     54153061
Mean Squared Error:  2.261908446988301e+16
Root Mean Squared Error:  150396424.39194825
MAPE :826.5616182370247
R2: -0.9317984212354609

55
Mean True Value:         84950034
Mean Absolute Error:     55005401
Mean Squared Error:  9147388791499650.0
Root Mean Squared Error:  95641982.369144
MAPE :1104.2638165971384
R2: 0.24535925719705243

8878
Mean True Value:         84309897
Mean Absolute Error:     54277374
Mean Squared Error:  9377959238697192.0
Root Mean Squared Error:  96839863.89239295
MAPE :1545.0408488673352
R2: 0.2049561655447404
```

In [28]:

```
import xgboost as xg
```

In [29]:

```
#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")
```

```
1541
Mean True Value:         84338182
Mean Absolute Error:     16088329
Mean Squared Error:  1138614692905361.2
Root Mean Squared Error:  33743365.1686574
MAPE :114.38197797167567
R2: 0.9045144804003565

5641
Mean True Value:         84332413
Mean Absolute Error:     16384662
Mean Squared Error:  1181244012528314.0
Root Mean Squared Error:  34369230.607162476
MAPE :56.22594013095263
R2: 0.8992227778496157

8977
Mean True Value:         84289999
Mean Absolute Error:     16252869
Mean Squared Error:  1157630148155565.8
Root Mean Squared Error:  34023964.32157143
MAPE :51.37917112564205
R2: 0.9015652082234624
```

In [ ]:

In [30]:

```
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")
```

```
5626
Mean True Value:         84067576
Mean Absolute Error:     32864432
Mean Squared Error:  3195103903233775.0
Root Mean Squared Error:  56525250.1386219
MAPE :808.6107414878769
R2: 0.7303356522831486

3795
Mean True Value:         84232117
Mean Absolute Error:     32258290
Mean Squared Error:  3102852439300382.0
Root Mean Squared Error:  55703253.39960299
MAPE :1428.7644701294769
R2: 0.737042053264592

454
Mean True Value:         84362852
Mean Absolute Error:     33421235
Mean Squared Error:  3298164123928292.0
Root Mean Squared Error:  57429644.992184065
MAPE :3748.0546185631597
R2: 0.7236418495764236
```

In [ ]:

In [ ]:

```
In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:
```