Capstone Project
SGEMM GPU Kernel Performance

Team: Data Defenders
Saraswat Mukherjee
Lokesh Tokas
Shubham Sartape
Charan

Contents

Problem Statement

- This data set measures the running time of a matrix-matrix product $A*B=C$, where all matrices have size 2048 x 2048, using a parameterizable SGEMM GPU kernel with 241600 possible parameter combinations. For each tested combination, 4 runs were performed, and their results are reported as the 4 last columns. All times are measured in milliseconds*.

- There are 14 parameter, the first 10 are ordinal and can only take up to 4 different powers of two values, and the 4 last variables are binary. Out of 1327104 total parameter combinations, only 241600 are feasible (due to various kernel constraints). This data set contains the results for all these feasible combinations.

Data Pipeline

Part 1: Download the Data and perform EDA, check for null values, outliers, statistical nuances, correlation, etc…

Part 2: Partition the Data randomly into train and test set using a good train/test split percentage.

Part 3: Design a Machine Learning model to calculate the average GPU run time, using various algorithms (Logistic Regression / Linear Regression / Ridge Regression / Lasso Regression / Decision Tree).

Part 4: Report accuracy / error metrics for train and test sets and finalize Machine Learning Model with best fit algorithm.

Data Summary

Attribute Information:

- Independent variables:
- MWG, NWG: per-matrix 2D tiling at workgroup level: {16, 32, 64, 128} (integer)
- KWG: inner dimension of 2D tiling at workgroup level: {16, 32} (integer)
- MDIMC, NDIMC: local workgroup size: {8, 16, 32} (integer)
- MDIMA, NDIMB: local memory shape: {8, 16, 32} (integer)
- KWI: kernel loop unrolling factor: {2, 8} (integer)
- VWM, VWN: per-matrix vector widths for loading and storing: {1, 2, 4, 8} (integer)
- STRM, STRN: enable stride for accessing off-chip memory within a single thread: {0, 1} (categorical)
- SA, SB: per-matrix manual caching of the 2D workgroup tile: {0, 1} (categorical)

Output:

- Run1, Run2, Run3, Run4: performance times in milliseconds for 4 independent runs using the same parameters. They range between 13.25 and 3397.08

EDA

EDA (Importing Libraries and Data Wrangling)

```
[ ]  import pandas as pd
     import numpy as np
     import statsmodels.api as sm
     import sklearn
     from sklearn.model_selection import train_test_split
     from sklearn.linear_model import LinearRegression,Lasso,Ridge
     from sklearn.tree import DecisionTreeRegressor
     from sklearn.metrics import r2_score
     import matplotlib.pyplot as plt
     import seaborn as sns
     import plotly.express as px
```

```
[ ]  #hiding warning after final edit
     import warnings
     warnings.filterwarnings('ignore')
```

Merging the 4 run columns into a single 'runtime' column with mean value of all 4 run columns.

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 241600 entries, 0 to 241599
Data columns (total 18 columns):
 #   Column     Non-Null Count   Dtype
---  ------     --------------   -----
 0   MWG        241600 non-null  int64
 1   NWG        241600 non-null  int64
 2   KWG        241600 non-null  int64
 3   MDIMC      241600 non-null  int64
 4   NDIMC      241600 non-null  int64
 5   MDIMA      241600 non-null  int64
 6   NDIMB      241600 non-null  int64
 7   KWI        241600 non-null  int64
 8   VWM        241600 non-null  int64
 9   VWN        241600 non-null  int64
 10  STRM       241600 non-null  int64
 11  STRN       241600 non-null  int64
 12  SA         241600 non-null  int64
 13  SB         241600 non-null  int64
 14  Run1 (ms)  241600 non-null  float64
 15  Run2 (ms)  241600 non-null  float64
 16  Run3 (ms)  241600 non-null  float64
 17  Run4 (ms)  241600 non-null  float64
dtypes: float64(4), int64(14)
memory usage: 33.2 MB
```

```
[ ]  data['runtime'] = data[['Run1 (ms)','Run2 (ms)','Run3 (ms)','Run4 (ms)']].mean(axis=1)
     data = data.drop(columns=['Run1 (ms)','Run2 (ms)','Run3 (ms)','Run4 (ms)'])
```

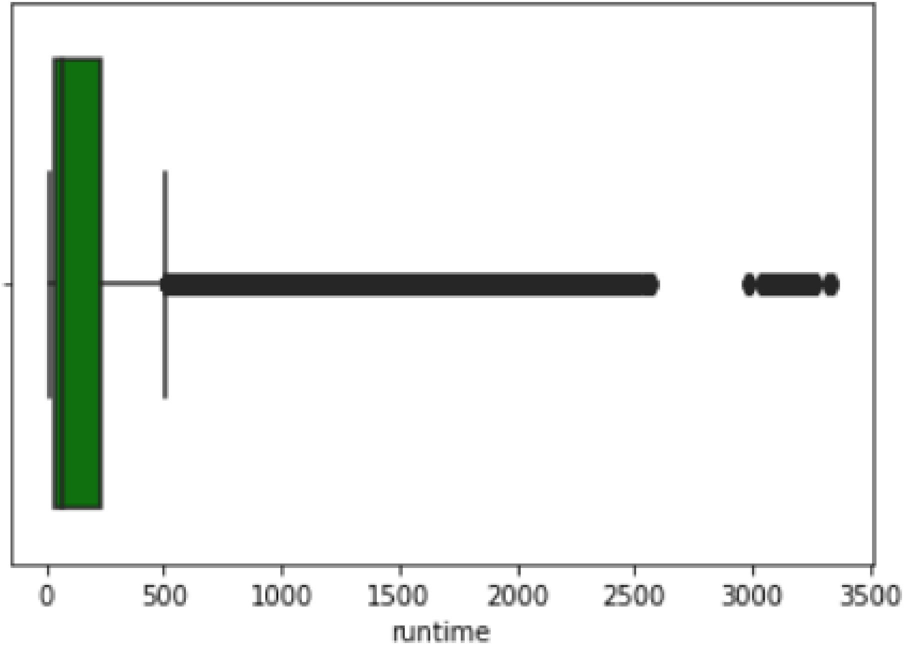
EDA (Checking Statistical Information)

```
[ ] data.describe().T
```

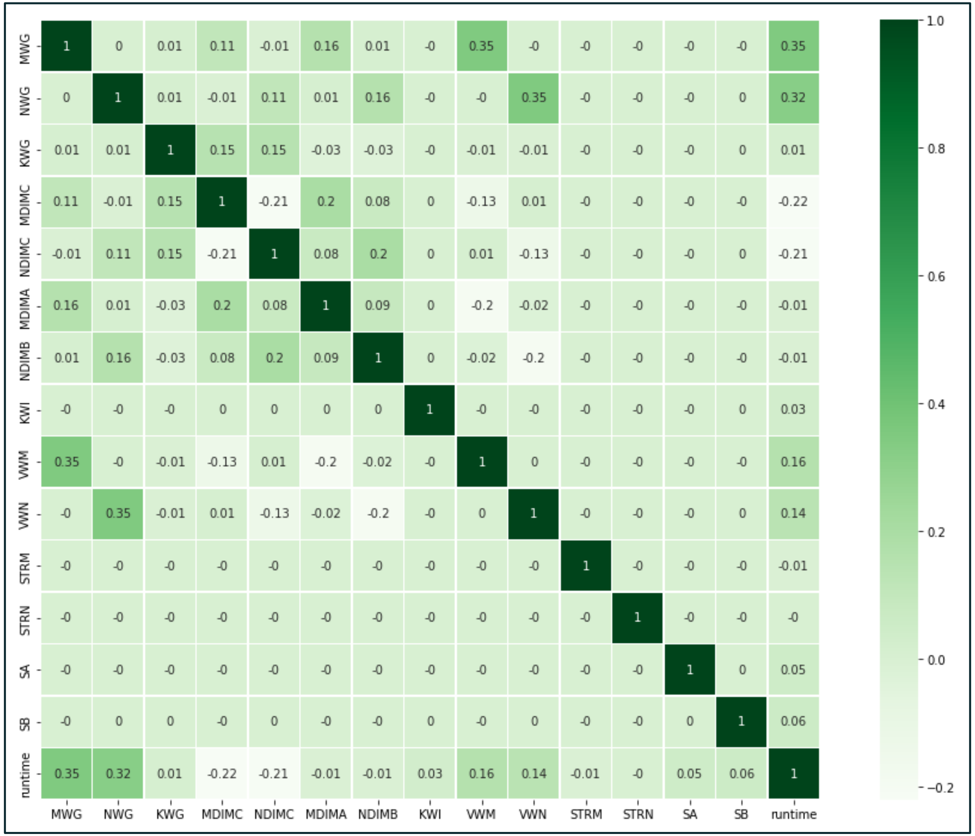
	count	mean	std	min	25%	50%	75%	max
MWG	241600.0	80.415364	42.469220	16.0000	32.0000	64.00	128.0000	128.0000
NWG	241600.0	80.415364	42.469220	16.0000	32.0000	64.00	128.0000	128.0000
KWG	241600.0	25.513113	7.855619	16.0000	16.0000	32.00	32.0000	32.0000
MDIMC	241600.0	13.935894	7.873662	8.0000	8.0000	8.00	16.0000	32.0000
NDIMC	241600.0	13.935894	7.873662	8.0000	8.0000	8.00	16.0000	32.0000
MDIMA	241600.0	17.371126	9.389418	8.0000	8.0000	16.00	32.0000	32.0000
NDIMB	241600.0	17.371126	9.389418	8.0000	8.0000	16.00	32.0000	32.0000
KWI	241600.0	5.000000	3.000006	2.0000	2.0000	5.00	8.0000	8.0000
VWM	241600.0	2.448609	1.953759	1.0000	1.0000	2.00	4.0000	8.0000
VWN	241600.0	2.448609	1.953759	1.0000	1.0000	2.00	4.0000	8.0000
STRM	241600.0	0.500000	0.500001	0.0000	0.0000	0.50	1.0000	1.0000
STRN	241600.0	0.500000	0.500001	0.0000	0.0000	0.50	1.0000	1.0000
SA	241600.0	0.500000	0.500001	0.0000	0.0000	0.50	1.0000	1.0000
SB	241600.0	0.500000	0.500001	0.0000	0.0000	0.50	1.0000	1.0000
runtime	241600.0	217.571953	368.750161	13.3175	40.6675	69.79	228.3875	3341.5075

Checking for Outliers in Runtime Column

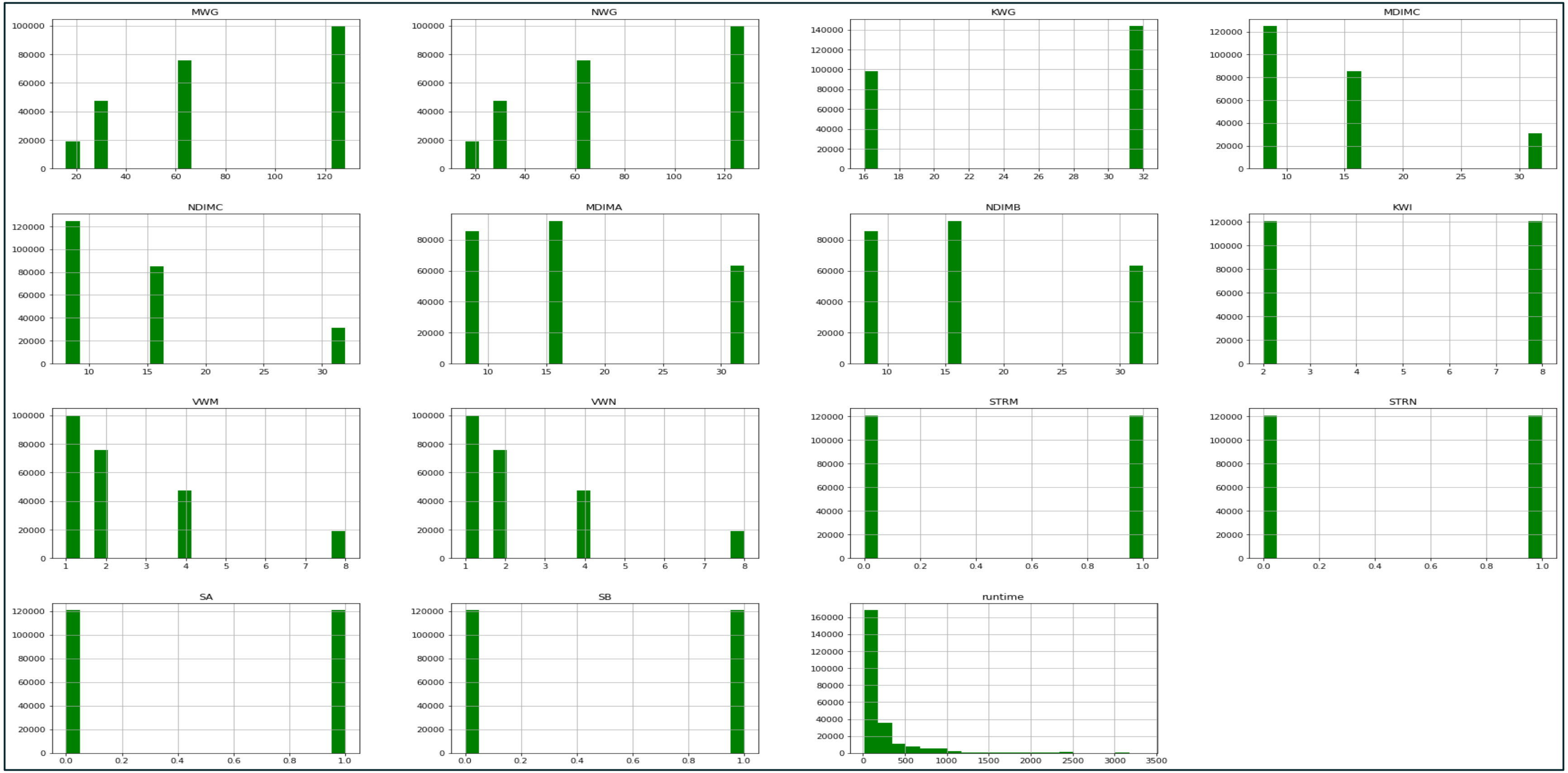
```
[ ]  sns.boxplot(data['runtime'],color='green')
     plt.show()
```



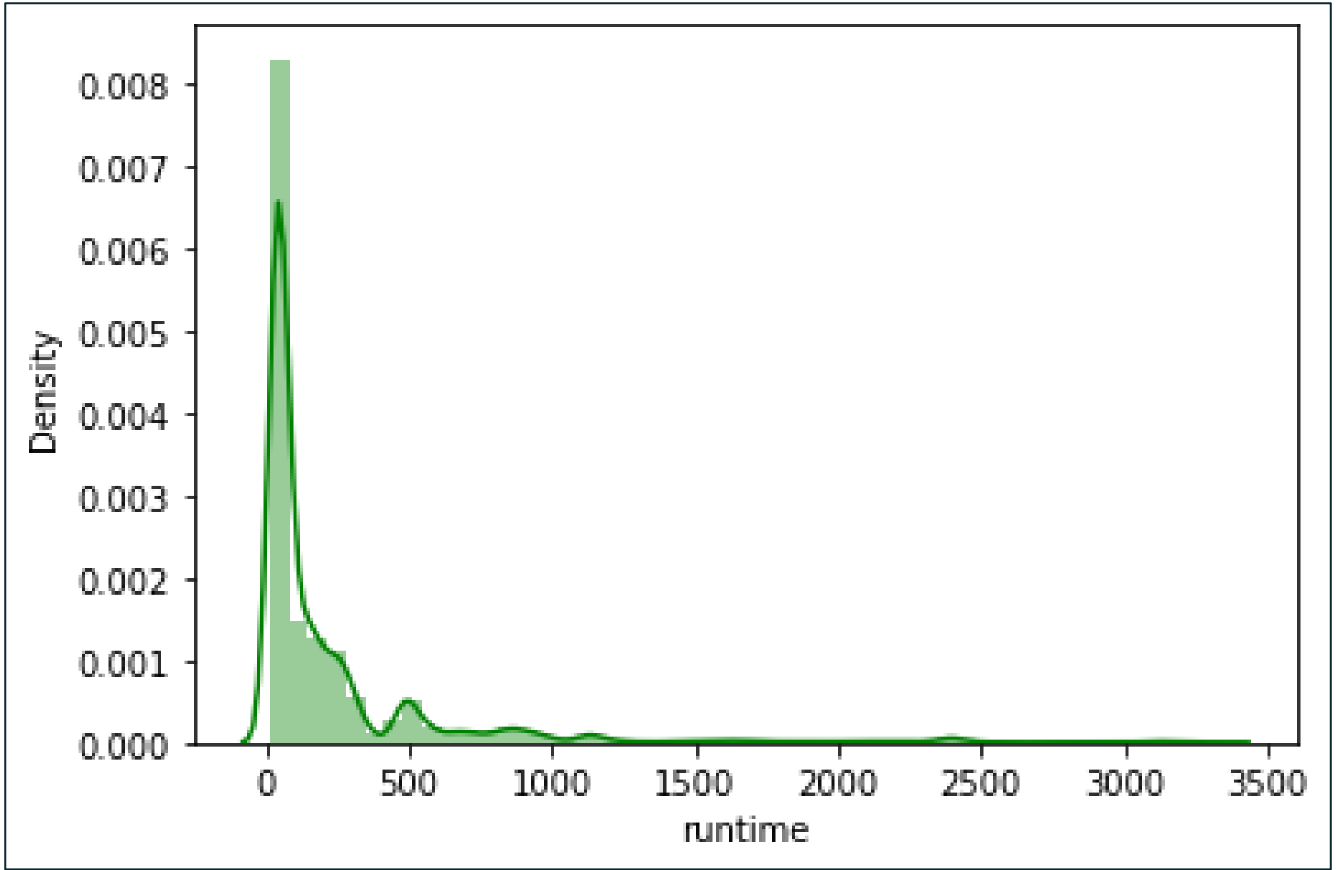
Heatmap Check for Data Correlation



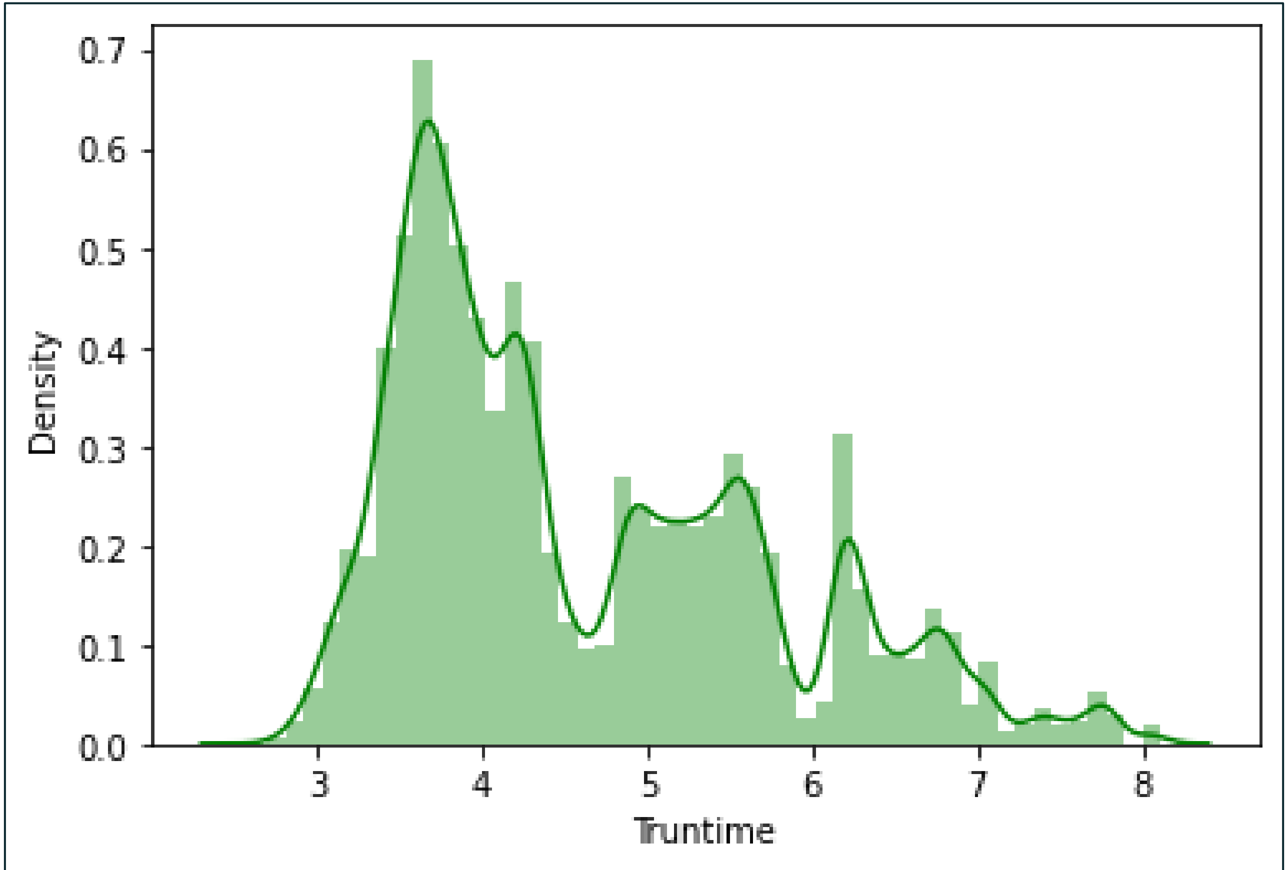
Creating Histogram for every Column



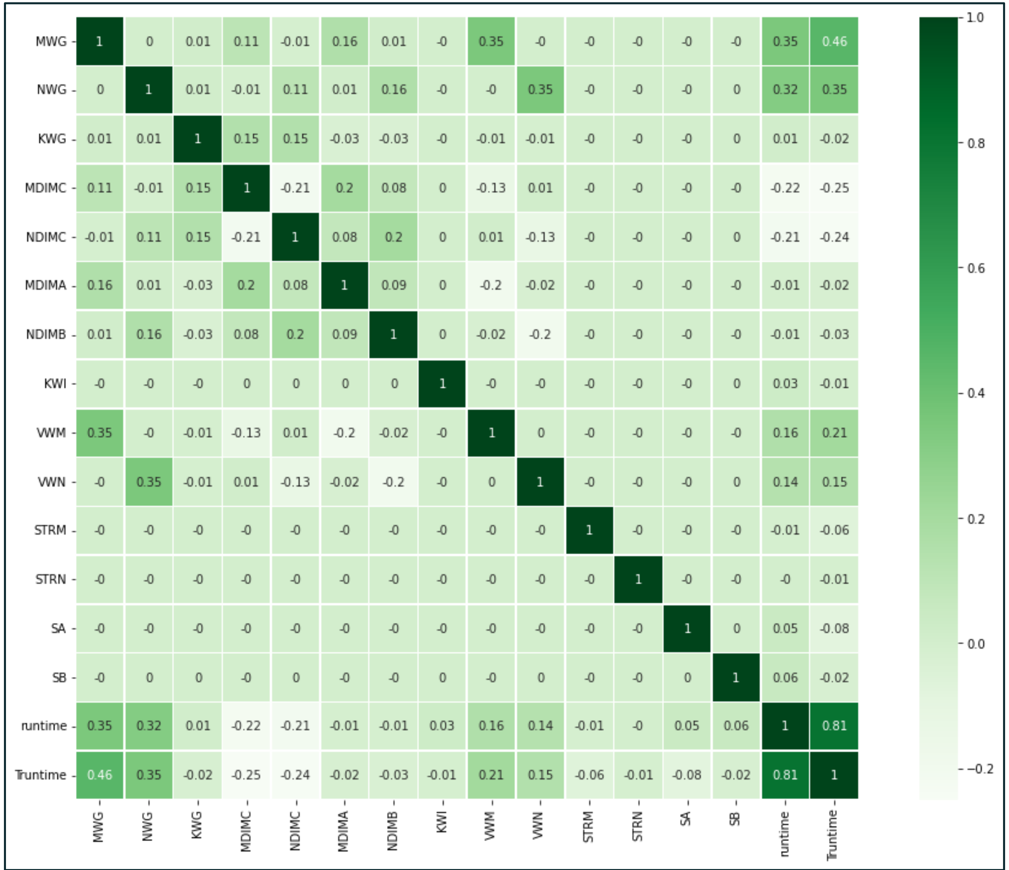
Checking Skewness of Runtime



Normalization of Runtime Data



Heatmap Check for correlation of Runtime Data



Machine Learning

Data for Machine Learning

```
[ ]  tdata = data.loc[:,data.columns != 'runtime']
     tdata.head()
```

	MWG	NWG	KWG	MDIMC	NDIMC	MDIMA	NDIMB	KWI	VWM	VWN	STRM	STRN	SA	SB	Truntime
0	16	16	16	8	8	8	8	2	1	1	0	0	0	0	4.756775
1	16	16	16	8	8	8	8	2	1	1	0	0	0	1	4.365707
2	16	16	16	8	8	8	8	2	1	1	0	0	1	0	4.389064
3	16	16	16	8	8	8	8	2	1	1	0	0	1	1	4.461733
4	16	16	16	8	8	8	8	2	1	1	0	1	0	0	4.776283

Test and Train Split

```
[ ]  X = tdata.loc[:, tdata.columns != 'Truntime']
     Y = tdata.loc[:, tdata.columns == 'Truntime']
     #test size limit set to 33%
     X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.33, random_state = 0)
```

```
[ ]  X_train.shape, X_test.shape, Y_train.shape, Y_test.shape

     ((161872, 14), (79728, 14), (161872, 1), (79728, 1))
```

R2 cal funtion

```
[ ]  def R2_cal(test,pred) :
       r2 = r2_score(Y_test, Y_pred)
       adj_r2 = 1-(1-r2)*((X_test.shape[0]-1)/(X_test.shape[0]-X_test.shape[1]-1))
       print(f"R2 Score : {r2}\nAdjusted R2 Score : ,{adj_r2}")
```

Linear Regression

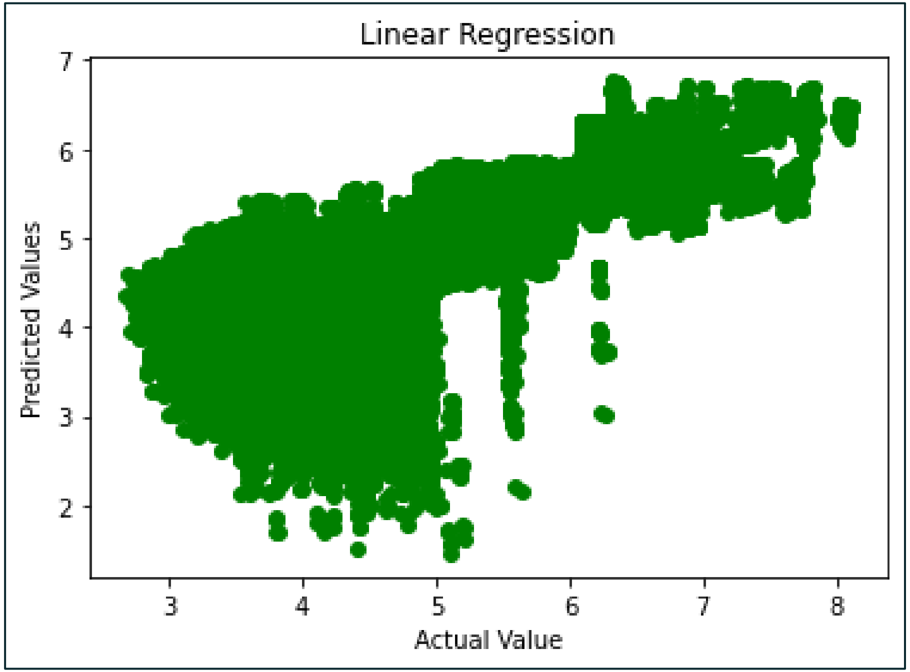
```
[22] lin_reg = LinearRegression()
     lin_reg.fit(X_train, Y_train)

     LinearRegression()

[23] lin_reg.intercept_

     array([4.22623861])

[24] lin_reg.coef_

     array([[ 1.33922435e-02,  1.05527302e-02,  1.26244894e-02,
             -5.68987338e-02, -5.47199485e-02,  9.17383153e-05,
             -1.30666224e-04, -3.81017128e-03, -8.91995503e-03,
             -2.36494387e-02, -1.34386687e-01, -1.64747066e-02,
             -1.88333908e-01, -4.80302852e-02]])

[25] Y_pred = lin_reg.predict(X_test)
     Y_pred

     array([[5.60704679],
            [4.36932169],
            [4.75638338],
            ...,
            [4.89167836],
            [4.68945049],
            [4.75061628]])
```

```
[26] df_pred = pd.DataFrame({'Actual' : Y_test.squeeze(), 'Predicted': Y_pred.squeeze()})
     df_pred
```

	Actual	Predicted
111345	6.730866	5.607047
62516	3.565298	4.369322
143068	4.220647	4.756383
152967	3.720257	4.804053
223400	6.205618	6.137559
...	...	...
225782	5.112966	5.164507
216435	4.837214	4.901438
163419	3.510052	4.891678
11190	3.738562	4.689450
135219	3.850201	4.750616

79728 rows × 2 columns

Linear Regression



R2 Score : 0.5574522121578815
Adjusted R2 Score : ,0.5573744874576471

Lasso Regression – Eliminating less Influential Features

```
[30] lasso = Lasso(alpha=0.001)
     lasso.fit(X_train, Y_train)
```

```
Lasso(alpha=0.001)
```

```
[31] Y_pred = lasso.predict(X_test)
     Y_pred
```

```
array([5.60223897, 4.3672886 , 4.75632054, ..., 4.89490594, 4.69029017,
       4.7493926 ])
```

R2-score & Adjusted R2-Score

```
[32] R2_cal(Y_test,Y_pred)
```

```
R2 Score : 0.5574366264776562
Adjusted R2 Score : ,0.5573588990401076
```

Ridge Regression – Checking Multicollinearity

```
[34] ridge = Ridge()
     ridge.fit(X_train, Y_train)

     Ridge()

[35] Y_pred = ridge.predict(X_test)
     Y_pred

     array([[5.60704303],
            [4.36931759],
            [4.75638232],
            ...,
            [4.89168262],
            [4.68945093],
            [4.75061727]])
```

R2-score & Adjusted R2-Score

```
[36] R2_cal(Y_test,Y_pred)

     R2 Score : 0.5574522050053216
     Adjusted R2 Score : ,0.5573744803038309
```

Decision Tree

```
[37] tree_reg = DecisionTreeRegressor()
```

```
[38] tree_reg.fit(X_train, Y_train)
```

```
DecisionTreeRegressor()
```

```
[39] Y_pred = tree_reg.predict(X_test)
     Y_pred
```

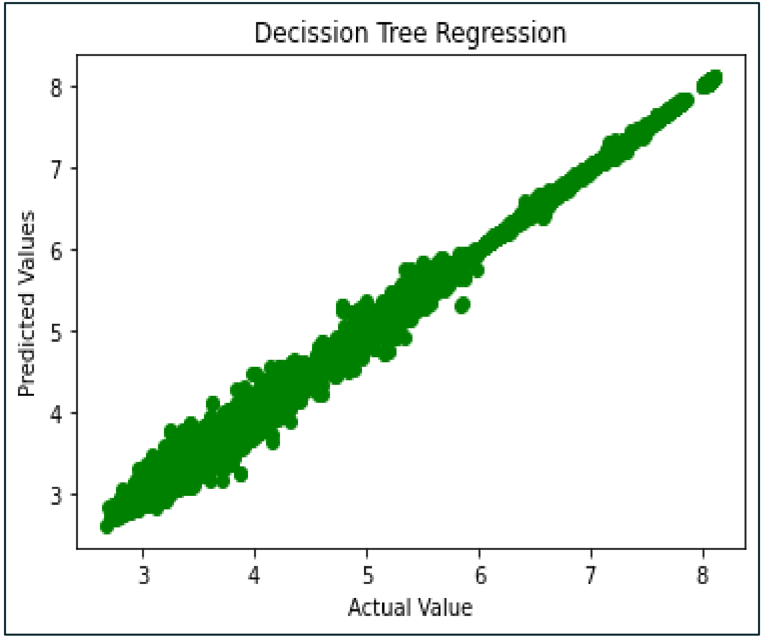
```
array([6.72420497, 3.56480322, 4.22218793, ..., 3.58004067, 3.65758227,
       3.73600156])
```

R2-score & Adjusted R2-Score

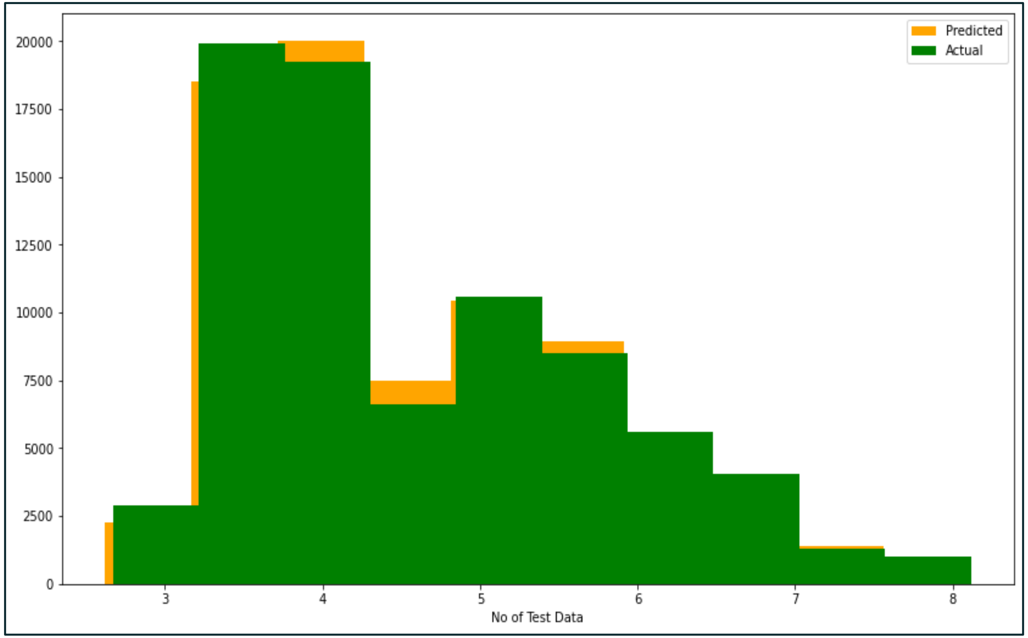
```
[40] R2_cal(Y_test,Y_pred)
```

```
R2 Score : 0.9988103101401573
Adjusted R2 Score : ,0.9988101011948405
```

Decision Tree

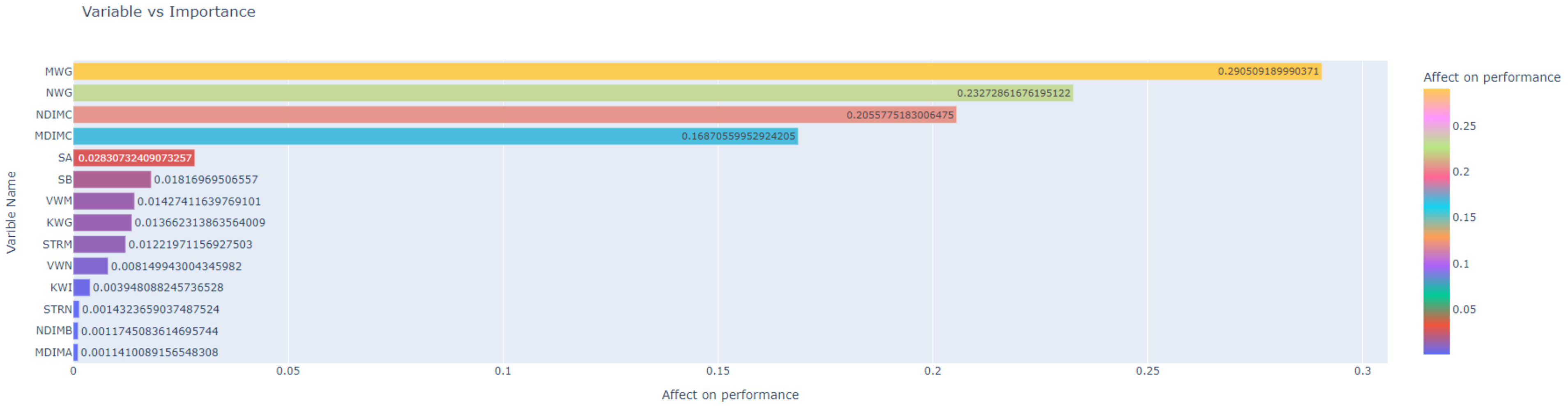


Checking Linearity Graphically



Checking Data Fit Graphically

Finding Features by their Importance



Findings

- Logarithmic Regression - Not applicable for this dataset since target variable is Float type

- Linear regression - Not suitable since adjusted R2 score = 0.55

- Lasso regression - Not suitable since adjusted R2 score = 0.55

- Ridge regression - Not suitable since adjusted R2 score = 0.55

- Decision Tree Regression - Ideal model with adjusted R2 score = 0.99

Conclusion

Variable	*Importance*
MWG	.290
NWG	.232
KWG	.013
MDIMC	.168
NDIMC	.205
MDIMA	.001
NDIMB	.001
KWI	.003
VWM	.014
VWN	.008
STRM	.012
STRN	.001
SA	.028
SB	.018

- Best Model depends on Factors such as R2 Score and Adjusted R2 Score
- The runtime of SGEMM GPU kernel is highly depending upon selection and values of all the parameters passed
- MWG, NWG, NDIMC and MDIMC majorly influence the runtime of the SGEMM GPU Kernel

Challenges

- Understanding the Problem Statement and Dataset.
- Finalizing and implementing the perfect Approach
- Execution and Delivering with Good Story Telling

Thank You