Capstone Project

SGEMM GPU Kernel performance prediction

by

Roshan Jamthe

Points for Discussion

- Problem Statement
- Data Summary
- EDA and Data Visualization
- Correlation Heatmap
- Train test split
- Linear Regression model
- XGBoost regressor model
- Neural Network model
- Best model
- Conclusion

Problem Statement

Predicting the running times for multiplying two 2048 x 2048 matrices using a GPU OpenCL SGEMM kernel with varying parameters (using the library 'CLTune').

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.

The experiment was run on a desktop workstation running Ubuntu 16.04 Linux with an Intel Core i5 (3.5GHz), 16GB RAM, and a NVidia Geforce GTX 680 4GB GF580 GTX-1.5GB GPU. We use the 'gemm_fast' kernel from the automatic OpenCL kernel tuning library 'CLTune' ([Web Link]).

Data Summary

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)

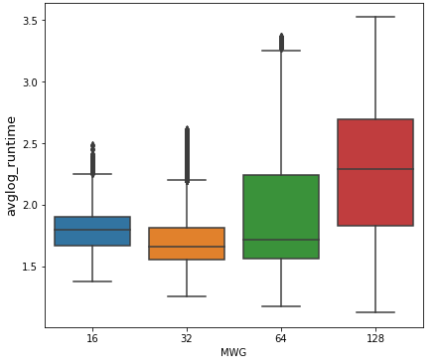
STRM, STRN: enable stride for accessing off-chip memory within a single thread: {0, 1} (categorical)

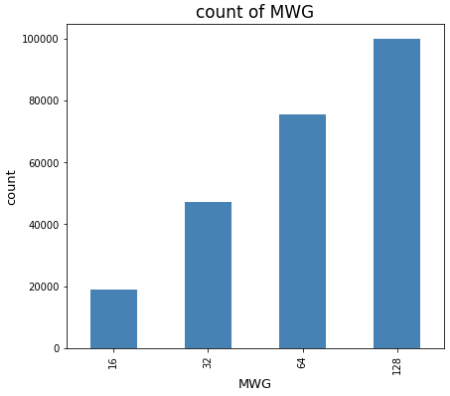
SA, SB: per-matrix manual caching of the 2D workgroup tile: {0, 1} (categorical)

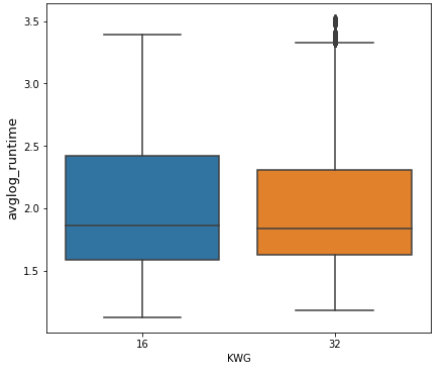
Output:

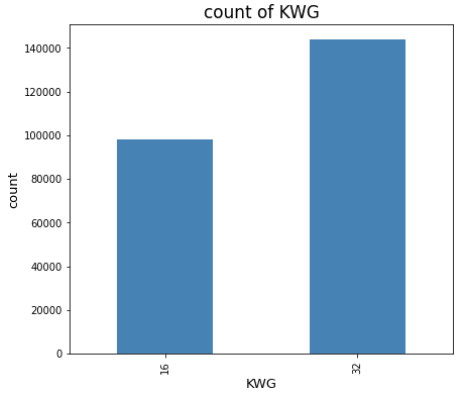
Run1, Run2, Run3, Run4: performance times in milliseconds for 4 independent runs using the same parameters. They range between 13.25 and 3397.08. (continuous)

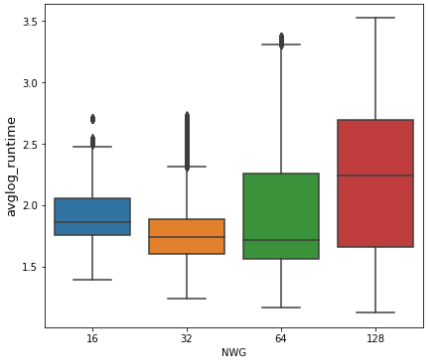
EDA and Data Visualization

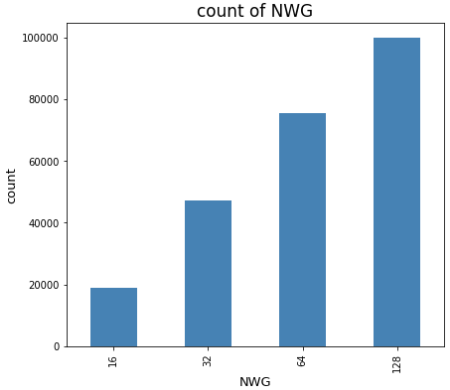


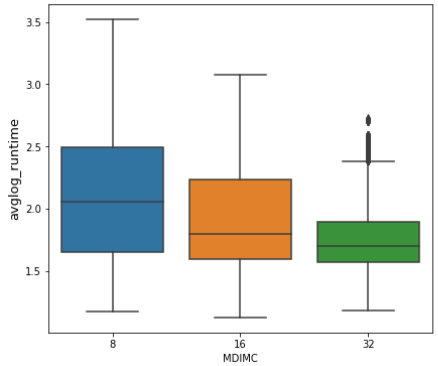


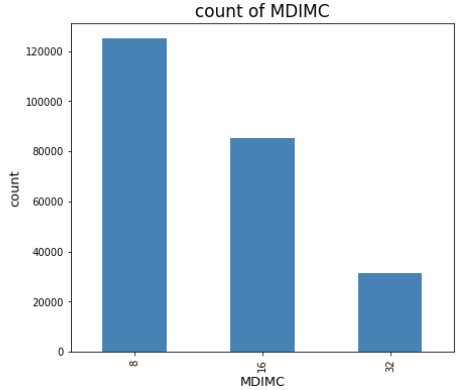




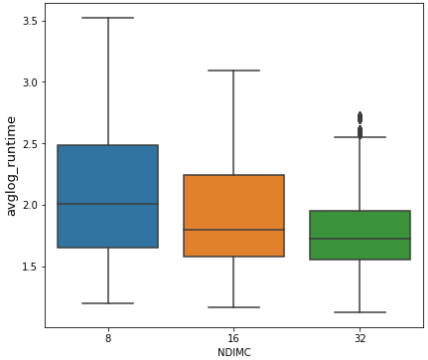


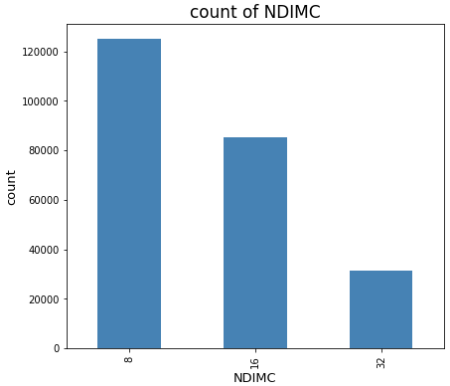


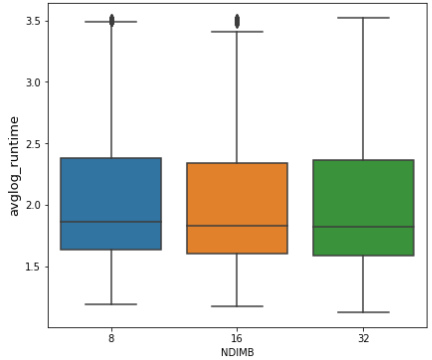


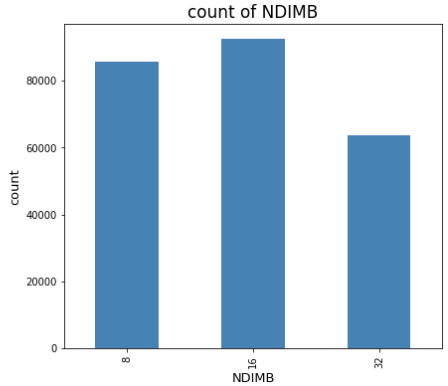


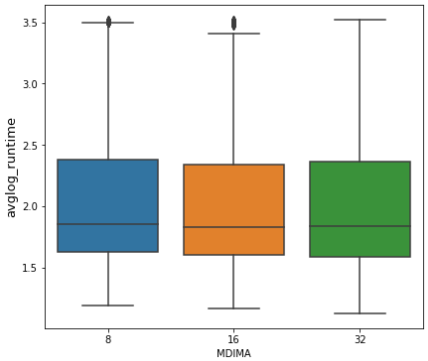
EDA and Data Visualization

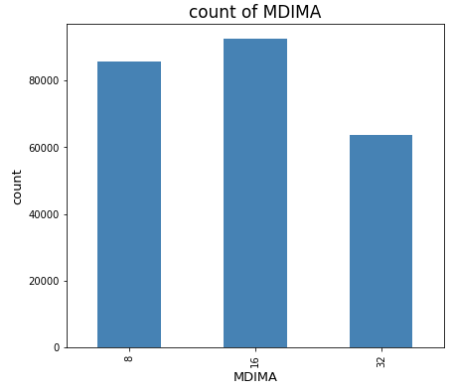


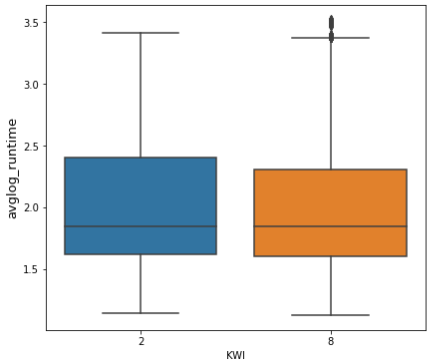


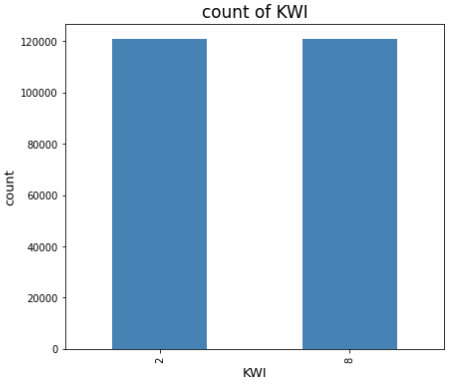




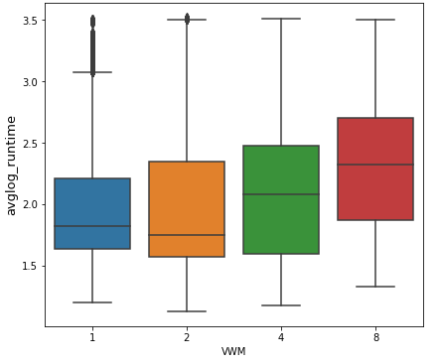


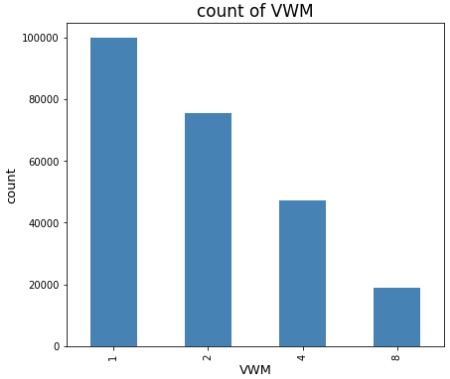


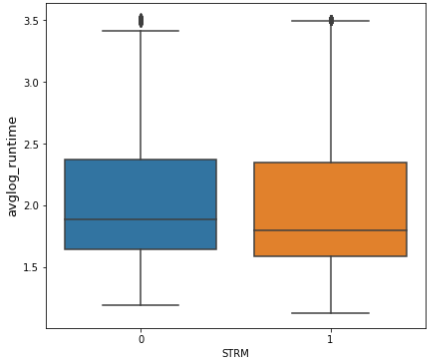


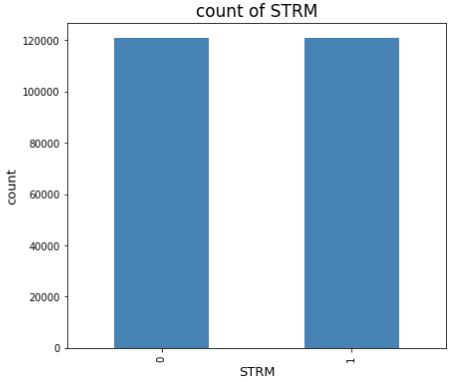


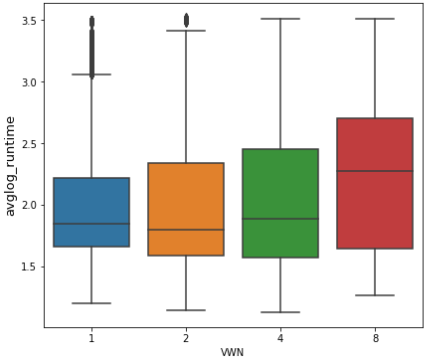
EDA and Data Visualization

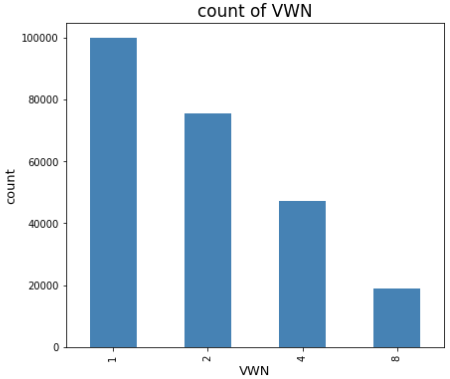


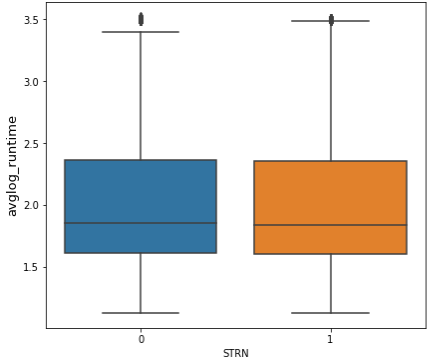


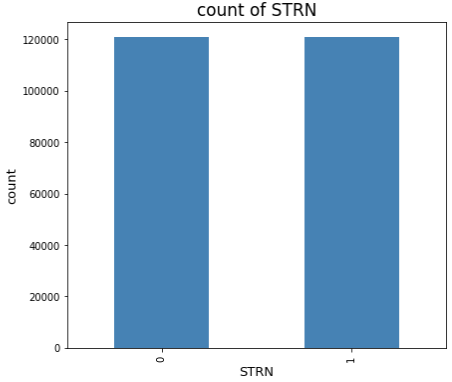




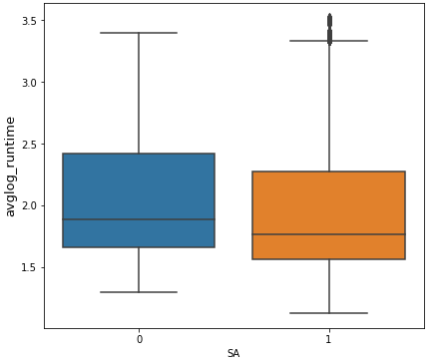


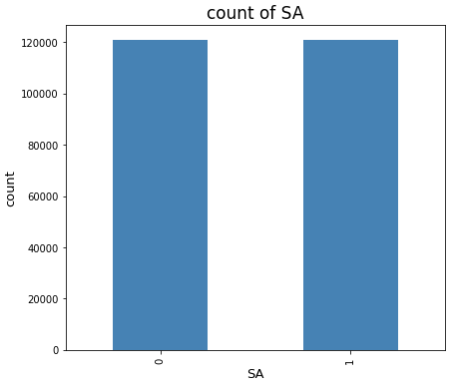


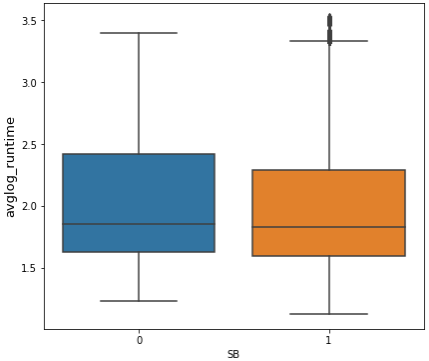


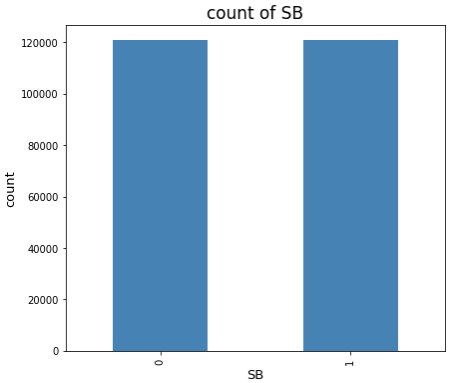


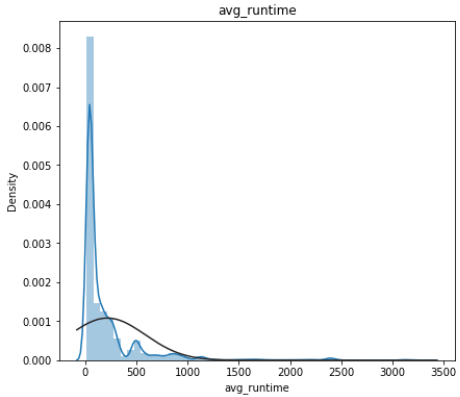
EDA and Data Visualization

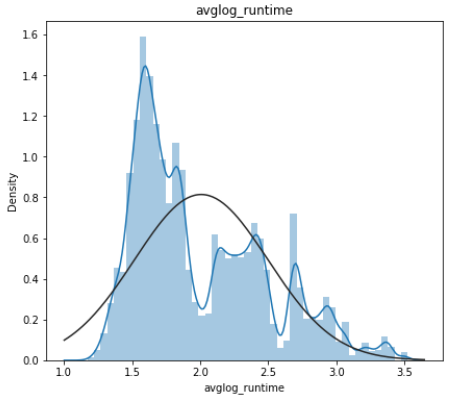


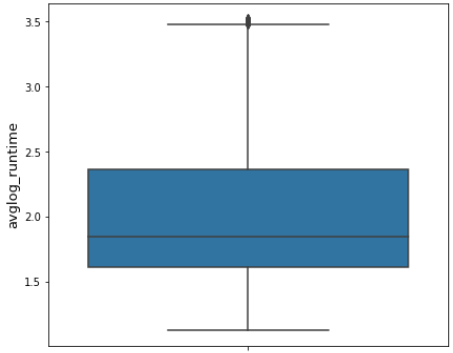


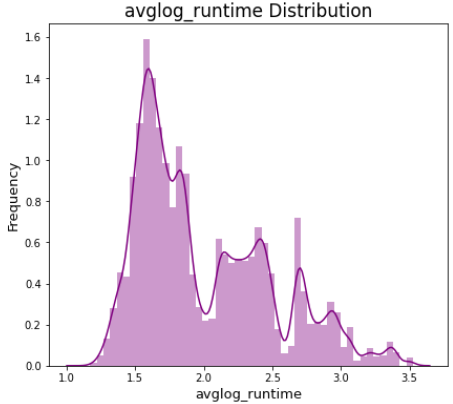




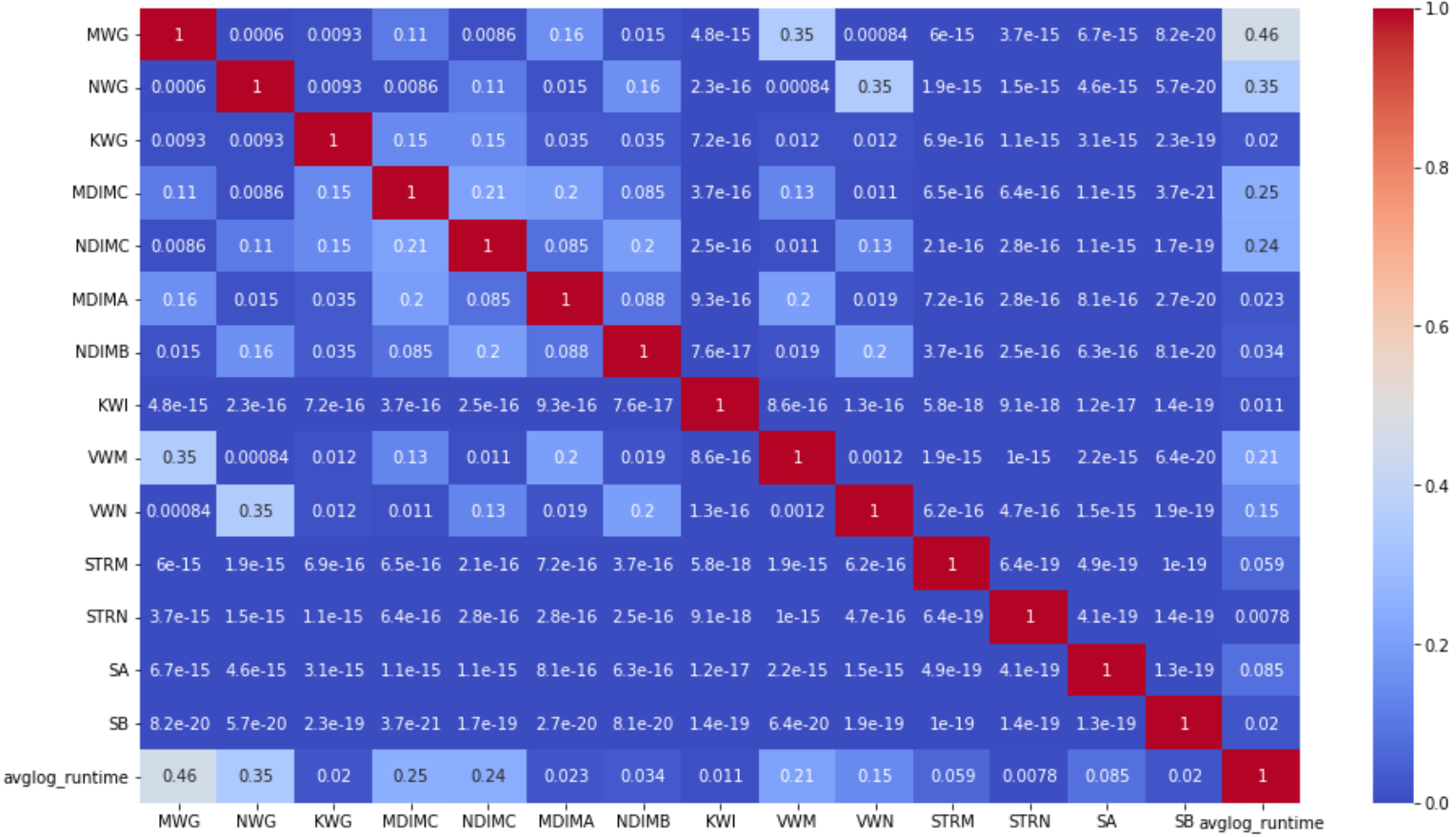






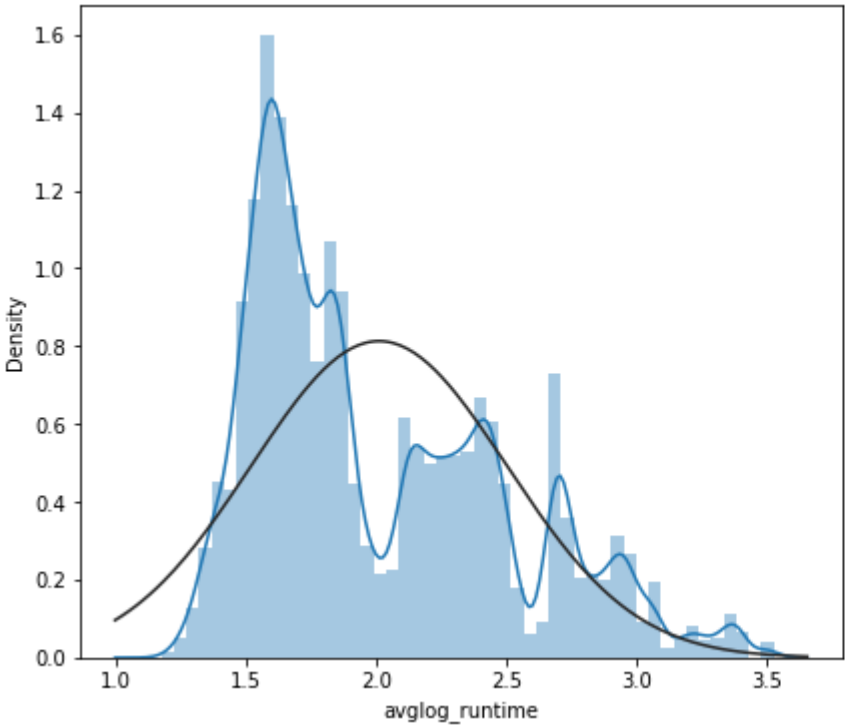


Correlation Heatmap

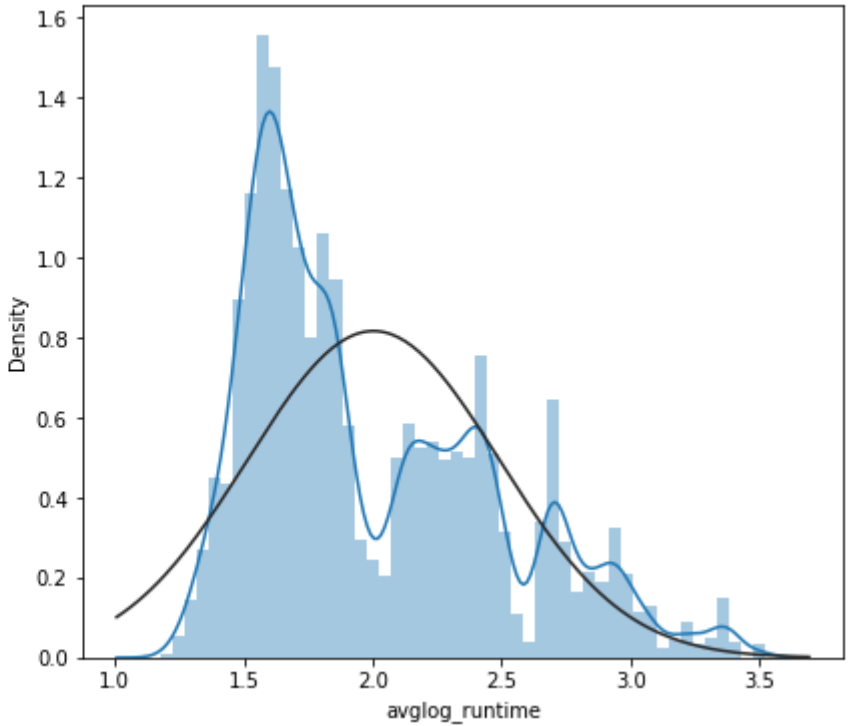


Train Test Split

Train dataset: 80% of the data

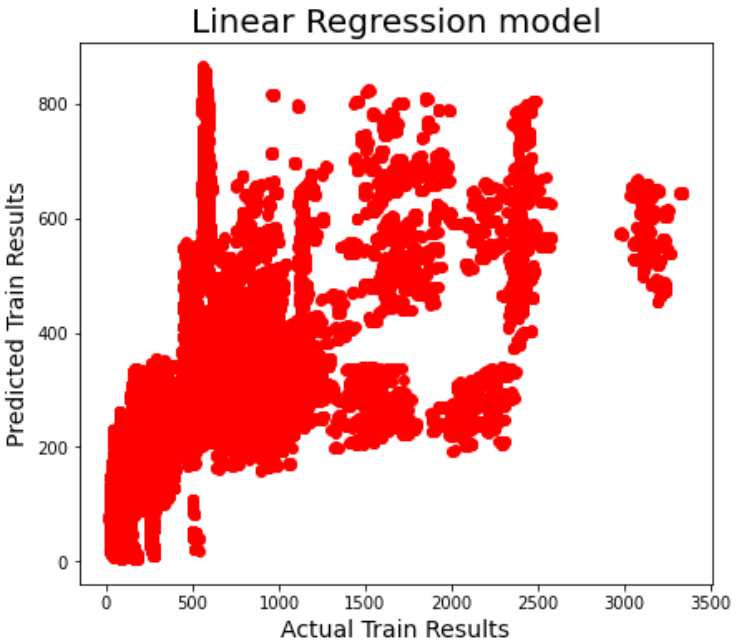


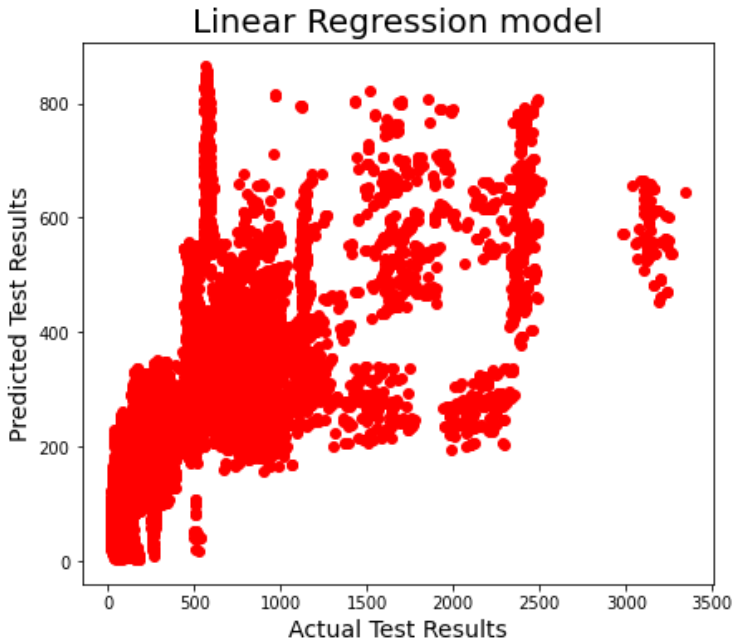
Test dataset: 20% of the data



Linear Regression model

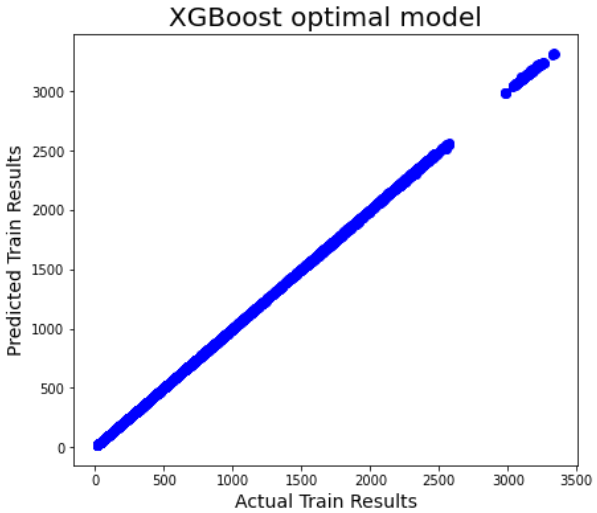
- To build a linear regression model, I used the sklearn library.
- **The model scored ~56% on both train and test datasets.** The results are compared below,

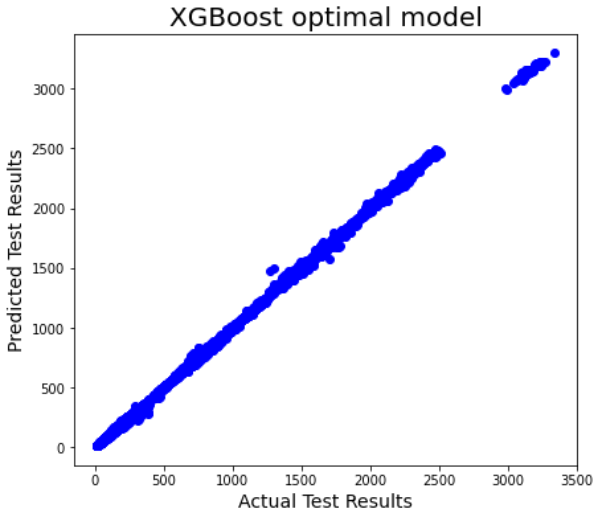




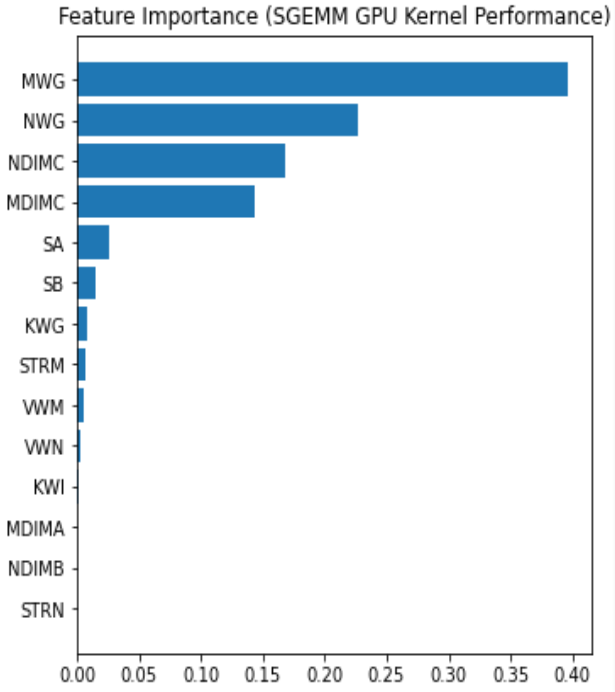
XGBoost Regressor model

- I employed BayesSearchCV to find the optimal value of max_depth
- **Optimal Model description :** n_estimators= 100, max_depth= 18, min_samples_split= 50, learning_rate= 0.1, loss= 'neg_mean_squared_error'
- **The model scored over 99.99% on both train and test datasets.** The results are compared below,





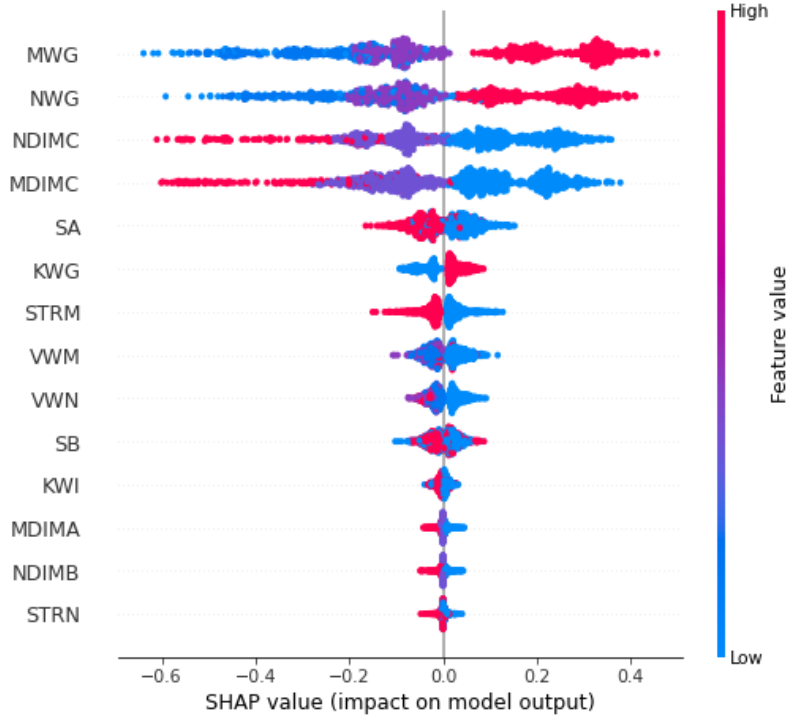
Model Interpretation : XGBoost Regressor model



Feature importance method

Weight	Feature
0.3959	MWG
0.2271	NWG
0.1676	NDIMC
0.1433	MDIMC
0.0257	SA
0.0149	SB
0.0086	KWG
0.0066	STRM
0.0053	VWM
0.0027	VWN
0.0010	KWI
0.0005	MDIMA
0.0005	NDIMB
0.0004	STRN

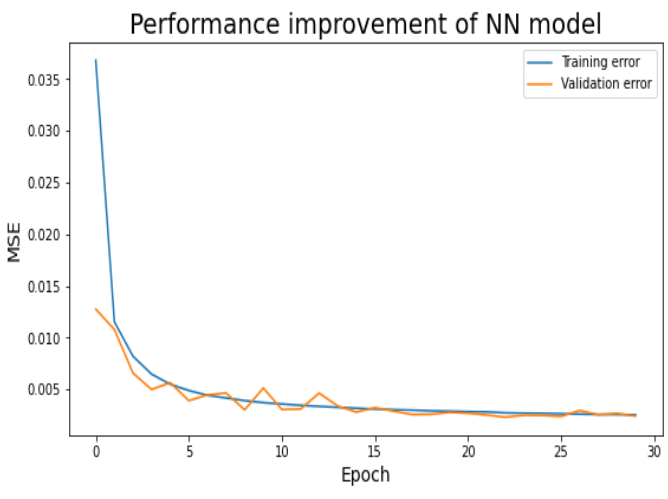
ELI5

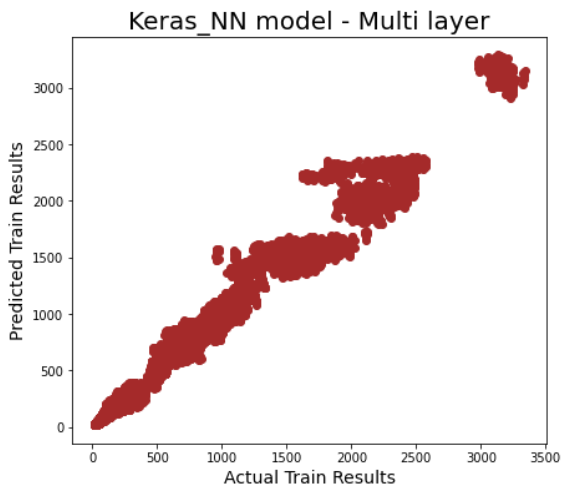


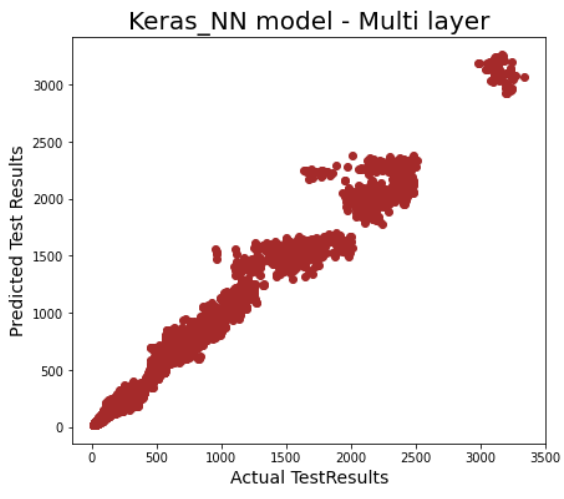
SHAP summary plot

Neural Network model

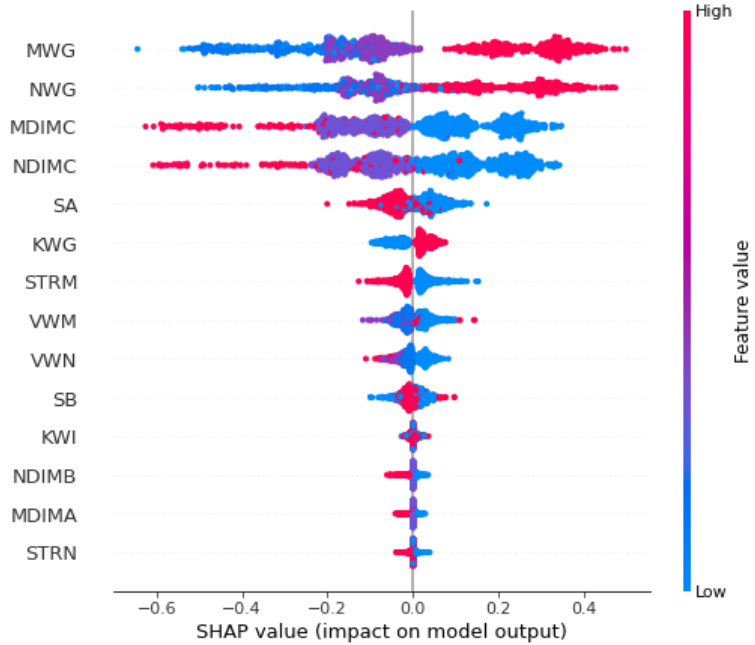
- An architecture of my multilayer neural network model build using keras library is, three dense layers having 64, 32, 8 neurons respectively with ‘relu’ activation function and one dense layer with 1 neuron with ‘linear’ activation function, loss='mean_squared_error', optimizer = adam, learning rate = 0.001.
- For training data, I used epoch = 30, batch size = 15, validation split = 0.2
- **The model scored ~98.00% on both train and test datasets.** The results are compared below,



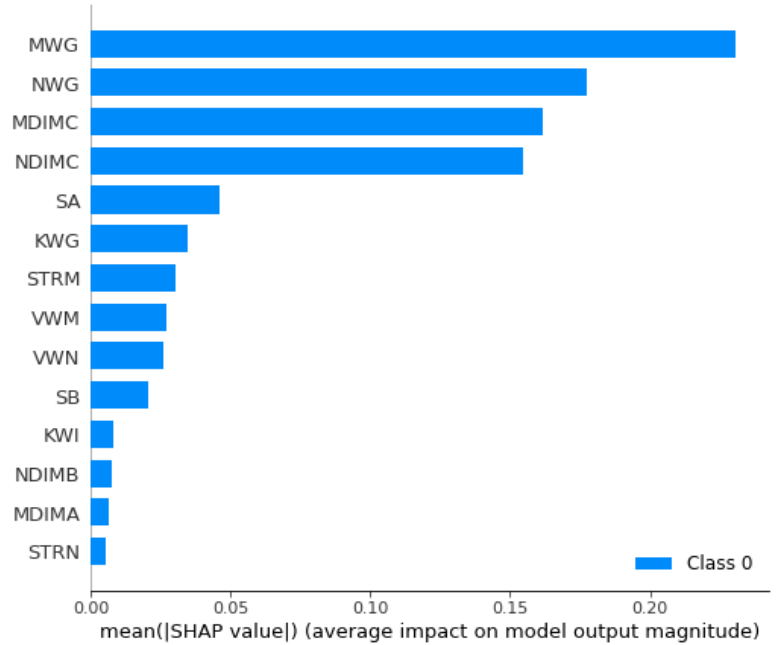




Model Interpretation : keras Neural Network model



SHAP summary plot



SHAP summary bar plot

Best model

- The 'Linear regression model' performed poorly because assumptions of linear regression didn’t get satisfied.
- The decision tree based model, 'XGBoost regressor' took higher building time with cross validation running, trying to find out best value for the parameter max_depth. But, the time was well spent.
- The 'Multi-layer Neural Network model' gave a slight generalized score than the XGBoost regressor model.
- Out of three, clearly, the XGBoost regressor model outperformed all the rest.

Model Name	Mean Squared Error (Train , Test)	Root Mean Squared Error (Train , Test)	R-square (Train , Test)
Linear regression model	(85135.749, 84026.006)	(291.78, 289.872)	(37.506, 37.736)
XGBoost regressor model	(1.633, 22.682)	(1.278, 4.763)	(99.999, 99.983)
Keras Neural Networks model	(2682.994, 2636.153)	(51.798, 51.343)	(98.031, 98.047)

Conclusions

In this study, so far I have done EDA and model building. I conclude that,

- The linear regression model was able to attain a poor score of ~56.00%.
- The multi-layer Neural Network model was able to attain a score of ~98.00% on both the train and test datasets.
- **The XGBoost regression model was able to attain a score better than 99.98% on both the train and test datasets. The error in predicting the runtime found to be less than 5 ms, a very accurate model since standard deviation of runtime feature is 368.75 ms.**

I was able to build an accurate runtime predictor model using XGBoost regressor. The feature interpretation from XGBoost and Neural network model indicate that,

- **MWG, NWG, NDIMC, and MDIMC contributed 90% in predicting runtime.**
- **MWG and NWG are positively correlated with runtime, while NDIMC and MDIMC are negatively correlated.**
- **The impact of MDIMA, NDIMB, and STRN on runtime is negligible.**

Thank you!