Randall Candaso
Professor Salena Ashton
ISTA 421
8 May 2025

# Final Project Paper

Final Project Github Link: https://github.com/randallcandaso/421_final_project

Base frequency serves as the default metric for CPU performance when a computer is running idle. It helps manufacturers test performance benchmarks, hardware design, and market forecasting. What determines a CPU's base frequency are the components used in its construction. Such components include: core count, thermal dynamic power, and lithography size. By utilizing the machine learning concept of regression prediction, a predictive model will be able to predict base frequency based on its architectural construction. This ability to predict base frequency can be useful for a manufacturer when trying to optimize production cost or for the consumer who is looking to buy the best performing processor for their money.

The dataset utilized in this project was taken from Kaggle.com and contains two main CSV files, one for AMD processors and the other for Intel processors. The AMD file contained twenty feature columns, while the Intel file contained twenty six. Of these features, were specific counts of the components previously mentioned. In addition to these, were many categorical columns of other components of CPU structure such as socket and memory type. When combined, the two files contained over fifteen hundred processor models that could be used for the project.

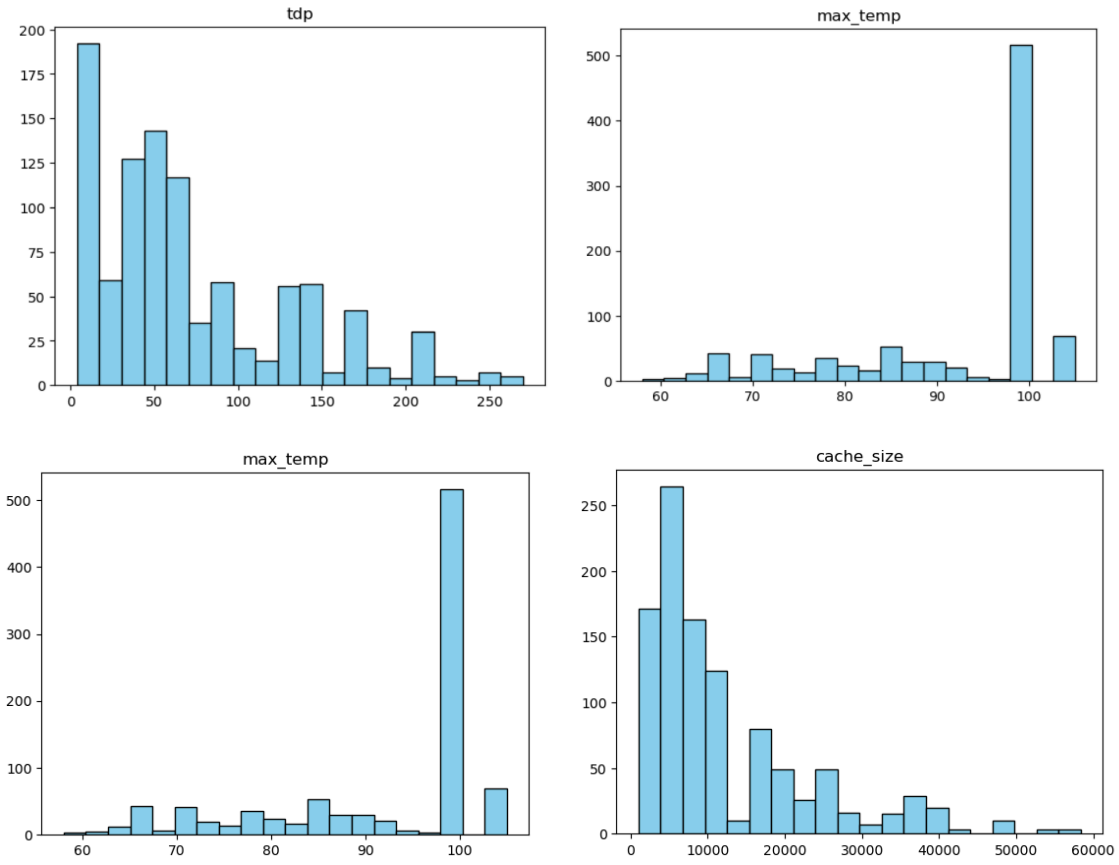
Before combining the two datasets however, much data exploration was done that led to an elaborate pre-processing stage. To start, it was necessary to determine which features could be used in the model that were present in both datasets. This caused a major barrier when arriving at both datasets' categorical columns. Entries for columns such as product line, socket type, and memory type had numerous unique values that needed to be simplified into a smaller number of groups. Additionally, because of Intel and AMD's differing architecture, unique component entries were not similar in both datasets. That made the simplification of categorical columns even more difficult in that groupings needed to be applicable to both datasets.

What was eventually decided upon, was to convert specific component product names to their use-case types for their respective components. For example, socket types were simplified into the types of sockets used in different types of computers such as desktop sockets, laptop sockets, or server sockets. For processor names, determining generations of CPU lines was difficult when having to apply to both manufacturer datasets and with the absence of enough release date entries, the performance tier of a processor was decided as the best categorization. Essentially, processor names were grouped into low, middle, and high tier models based on their contextual purposes and how they were promoted for use.

Next, through correlation analysis, columns were eliminated because of their high correlation with other features that also related contextually, such as turbo frequency with base frequency and thread count with core count. As mentioned before, columns with a smaller amount of entries such as release date, were dropped as well.

| | id | cores | threads | lithography | base_frequency | turbo_frequency | tdp | max_temp | max_memory_speed | sku | cache_size |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **id** | 1.000000 | -0.041299 | -0.146747 | 0.821001 | -0.258062 | -0.483501 | 0.077498 | -0.696202 | -0.642203 | -0.671938 | -0.084560 |
| **cores** | -0.041299 | 1.000000 | 0.988316 | -0.189585 | -0.120763 | -0.022651 | 0.756976 | -0.131554 | 0.404473 | 0.197813 | 0.874821 |
| **threads** | -0.146747 | 0.988316 | 1.000000 | -0.365189 | -0.101334 | -0.003975 | 0.733949 | 0.084761 | 0.470114 | 0.281622 | 0.889497 |
| **lithography** | 0.821001 | -0.189585 | -0.365189 | 1.000000 | -0.170470 | -0.441164 | 0.065003 | -0.783276 | -0.854282 | -0.772595 | -0.331490 |
| **base_frequency** | -0.258062 | -0.120763 | -0.101334 | -0.170470 | 1.000000 | 0.726367 | 0.277045 | -0.227909 | 0.352929 | 0.007811 | -0.029378 |
| **turbo_frequency** | -0.483501 | -0.022651 | -0.003975 | -0.441164 | 0.726367 | 1.000000 | 0.110947 | 0.066193 | 0.458997 | 0.277745 | 0.023005 |
| **tdp** | 0.077498 | 0.756976 | 0.733949 | 0.065003 | 0.277045 | 0.110947 | 1.000000 | -0.515818 | 0.375286 | -0.021863 | 0.757044 |
| **max_temp** | -0.696202 | -0.131554 | 0.084761 | -0.783276 | -0.227909 | 0.066193 | -0.515818 | 1.000000 | 0.562532 | 0.704001 | 0.054726 |
| **max_memory_speed** | -0.642203 | 0.404473 | 0.470114 | -0.854282 | 0.352929 | 0.458997 | 0.375286 | 0.562532 | 1.000000 | 0.647568 | 0.493183 |
| **sku** | -0.671938 | 0.197813 | 0.281622 | -0.772595 | 0.007811 | 0.277745 | -0.021863 | 0.704001 | 0.647568 | 1.000000 | 0.334363 |
| **cache_size** | -0.084560 | 0.874821 | 0.889497 | -0.331490 | -0.029378 | 0.023005 | 0.757044 | 0.054726 | 0.493183 | 0.334363 | 1.000000 |

Once the set of variables was determined applicable for both manufacturer datasets, it was time to fill in missing data for our numerical columns. The skew behavior of each numerical column was both plotted and calculated for, to determine how each column's missing data would be filled. Provided below will be some of the skew plots that were found, but the general rule used was a high skew behavior would require the median to fill the value or if a smaller skew behavior appeared, the mean would be used instead. After all the preprocessing was done, the two datasets are combined into one main dataset due to, at this point of the partnered code, the two datasets having the same columns, column order, and column-categories.




In terms of determining the right machine learning model for this scenario, there are a number of requirements that need to be met. Because the model will be predicting on base frequency, which happens to be numeric, the model will need to be one of regression. Additionally, because prediction is based upon processor components, an interpretable model is needed to be able to analyze the strength of influence each component has on base frequency. Lastly, the data utilized is highly complex in nature. There are a number of both numerical and categorical features for over fifteen hundred instances. Of these categorical features, multiple dummy columns will need to be created for each feature, to hot encode each individual category type it holds. The model that will be used will have to not only be able to handle this feature complexity but it will also be helpful if it is able to either penalize feature influence or completely feature-select altogether.

By using this model criteria, it was decided that a Lasso Regression model would be the best fit model of choice. What is most unique about Lasso is not just its ability to penalize feature influence, but being able to zero out features completely, if determined not to be influential enough. This behavior categorizes it as a regularization technique and with

regularization comes avoiding overfitting. Because of Lasso's ability to penalize, this prediction model will not just be stable in its predictions but when paired with cross-validation, it will generalize itself to become replicable in performance for future use. Additionally, due to Lasso being a form of linear regression, the model will be able to provide the coefficients used in its calculations which will allow for feature analysis.

After creating the code for a scratch-made Lasso model, another model was also created from the "sklearn" package to use as a comparison for results. To start, five-fold cross-validation was utilized in training to produce the most optimal alpha penalty value for each respective model. An alpha value of one was determined for the package model after validation, in comparison to an alpha value of fifty for the scratch. This process of determining alpha values helps generalize the models so that they don't just perform well with the given dataset, but with future data as well.

After determining these alpha values, they were then entered as parameters into a new version of their respective model and trained off of eighty percent of the data. Following this fitting, the models were then used to predict on the base frequency test split. In terms of MSE, the package model calculated a value of 158545.32, compared to the scratch model value of 211920.21. To make interpretation easier, the RMSE values calculated to 398.18 for the package and 460.35 for the scratch. This means that the scratch model's predictions were off about 460 megahertz in terms of base-frequency. When considering that base frequency can range between fifteen hundred to over three thousand megahertz in the dataset, the model's average error is not too far off with the given context.

The average error of the scratch model begins to look better when taking into account the model's r-squared value. For the package model, it explained about seventy-two percent of variance while the scratch model accounted for about sixty two percent. These r-squared values indicate that the models capture a substantial portion of the complexity within the data. In addition to this, because they were trained using techniques such as cross-validation and regularization, their performance is not just limited to the training set, but to new, unseen data as well.

| | Package | Scratch |
|---|---|---|
| **cores** | -649.792734 | -302.931578 |
| **lithography** | -190.099479 | -19.958542 |
| **tdp** | 655.9192 | 475.872158 |
| **max_temp** | -0.0 | 0.0 |
| **sku** | -7.490713 | 0.0 |
| **cache_size** | 157.00913 | 0.0 |
| **brand** | -188.708053 | -125.254506 |
| **processor_performance_Low Tier** | 98.075066 | 0.0 |
| **processor_performance_Mid Tier** | 47.849561 | 0.0 |
| **processor_performance_Other** | -32.430284 | 0.0 |
| **socket_group_FCPGA** | 98.390505 | 51.557664 |
| **socket_group_LGA** | 143.024184 | 148.846328 |
| **socket_group_Other** | -16.49161 | -14.684338 |
| **socket_group_Threadripper** | 15.794784 | 14.837418 |
| **memory_group_DDR3** | 0.0 | 0.0 |
| **memory_group_DDR4** | 27.548419 | 0.0 |
| **memory_group_LPDDR4** | -35.974375 | 0.0 |
| **memory_group_Other** | -35.567382 | -59.766491 |

Above is a table containing the coefficient values of the feature set for both the scratch and package models. While the general performance of each model was discussed, this table allows for further analysis of the given model choice and highlights the purpose of interpretability. At first glance, the most obvious aspect of the table is the zeroed features that each model deemed uninfluential. Just maximum temperature and the DDR3 memory type were zeroed out for the package. Comparatively, in addition to the two previous features mentioned, features such as cache size and other memory types were zeroed out in the scratch model.

In terms of having the most influence on base frequency, the number of cores a CPU contains held the greatest coefficient for both models. This makes sense because of the nature of increasing core count that decreases the allowed base frequency within each core. The second highest coefficient for both models was for thermal dynamic power. Thermal dynamic power

essentially functions as the available power bank of a CPU in which it is able to draw from. The greater TDP, the more a CPU can perform.

A noticeable difference between the two models is the emphasis of lithography size that is present in the package model but shown to have little influence in the scratch. This speaks to the room for improvement in the scratch-made model in its inability to account for this relationship. Lithography refers to the size of a CPU's transistors that are present on a chip. The smaller the transistor size, the greater number you can fit onto a chip and with more transistors comes better performance overall.

Pairing the difference in numerous features' influence with the higher number of zeroed-out columns, there is evidence of a difference in data interpretation that the models are doing. Because the package model's creation does not thoroughly illustrate the mathematics behind its performance, there is a limitation in the ability to compare the internal decision-making processes of both models. However, along with the differing alpha values, the resulting sparsity suggests that each model uses a different threshold for determining which features are influential. This comparison highlights how the aspects of implementation, such as penalty strength and optimization calculation, can significantly impact a model's interpretability and feature selection.

While the scratch model did not perform one-to-one with its "sklearn" counterpart, its performance metrics illustrate that the model is still usable for the goals of this dataset. As mentioned before, the model was off about 460 megahurtz when predicting base frequency and was able to still explain sixty two percent of the data's complexity. While not perfect, it performed as a reliable model for not just predicting base frequency, but also in explaining some of the component relationships with the target variable.

Components such as core count, thermal dynamic power, and socket type appeared as highly influential features in predicting base frequency. The model's r-squared value reinforces the reliability of these relationships, indicating that the model captures a substantial portion of the variance in base frequency based on its input features. This gives confidence that this feature influence is not simply overfitting to the training data, but is generalizing enough to unseen processor configurations.

In further implementation or dataset expansion, the model's provided context helps when dealing with deeper feature engineering. This can be in the form of better-detailed classification

of some categorical components such as socket or memory type. Another method can be adding additional information such as manufacturing year and voltage power to the dataset. With a feature such as core count, handling the relationship between core count and thread count can be made easier with the Lasso model's context as well.

Overall, this dataset project allowed me to exercise the abilities essential to standard data analysis and machine programming. By taking a complex dataset such as the processor files received from Kaggle, practices of data exploration and pre-processing were required to create a dataset clean enough to use in a machine learning model. This required vast exploration of unique entries and categorizations, feature correlations, and imputing missing values. After pre-processing, came the step of model selection which required the knowledge of varying machine learning models and their use-cases. By analyzing the problem statement decided upon from the dataset, a model choice was made that fit the purpose of the project. A scratch-made model was then created from hand-code, allowing for further understanding of optimization and regularization techniques that are crucial in the real-world application of predictive modeling.

From a broader perspective, this project enhanced the communicative skills that are vital in both graduate level research and real-world industry roles. Being able to ask data-driven research questions, interpret modelling outputs, and communicate analysis effectively are all crucial in collaborative environments that surround data analysis. In this project, the ability to draw meaningful conclusions from my quantitative results and frame them in the proper context has prepared me to contribute meaningfully in future collaborative spaces and more complex data projects.