Chapter Two – The Music Project

Our initial motivation to take on the Music project was a general interest in music. This type of problem seemed very interesting in particular because of the ‘unknown’ nature of the features in the dataset. Music is very mathematical in nature which really got us thinking of ideas to combine variables in ways that formed mathematical patterns commonly found in music. However, most of those ideas made our models worse.

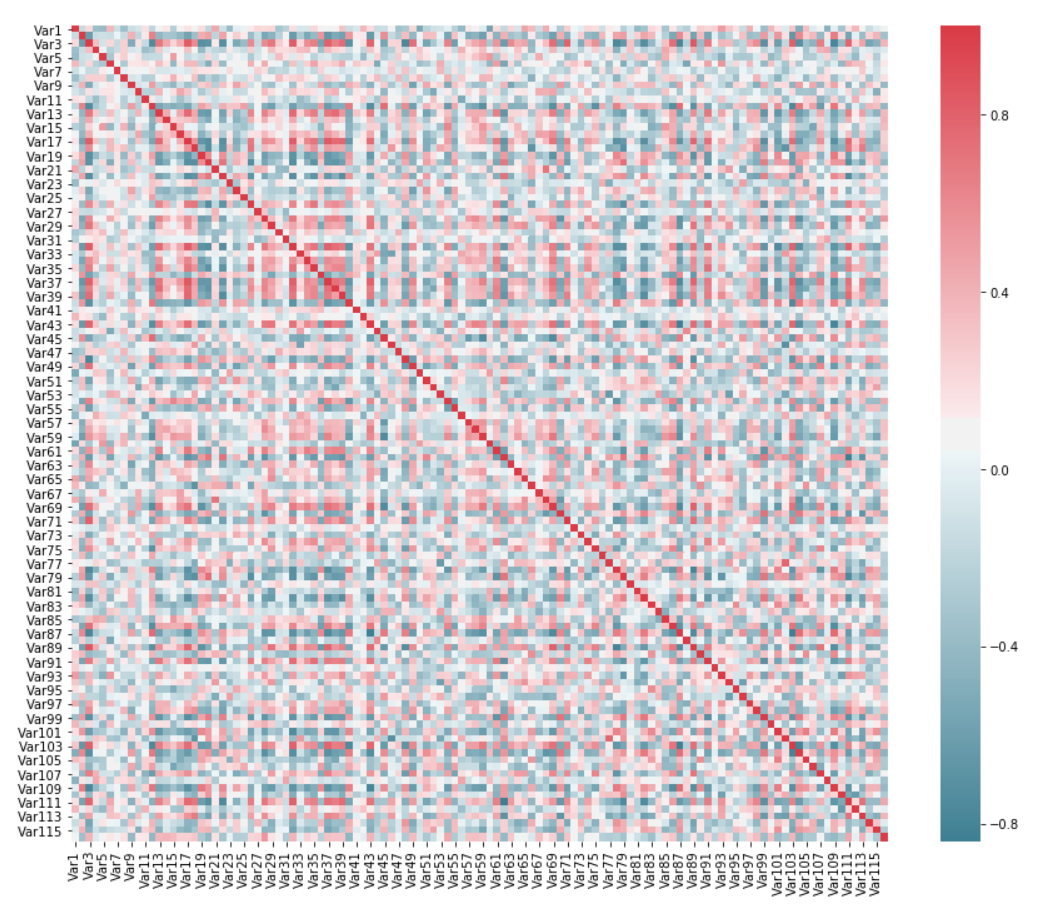
We implemented our project in Python. The following section will describe each type of model implemented for our project, the motivation behind them, and an outline of our process.

**Model 1: Linear Regression**

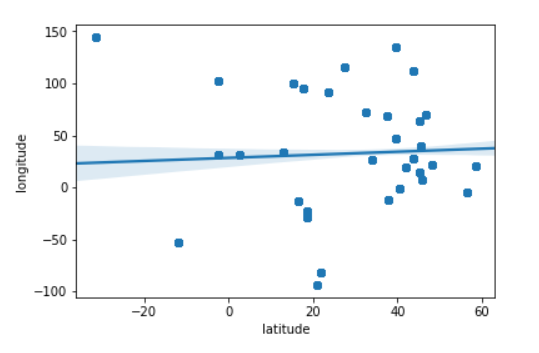
The dataset for this project consisted of 959 observations with 118 columns. Columns 1-116 were named Var1-Var16, respectively, and represented sound and wave synthesis features extracted from music clips around the world. Column 117 and 118 were latitude and longitude coordinates. All features in the dataset were numeric in nature. Since this dataset had a low volume of observations and a high count of features, the predictions were generally not easy to make.

We first started by pulling in the dataset, converting it to a data frame for easier handling, and splitting it into feature variables and a response variable. The features Var1 through Var116 were assigned to a variable “X”. The response variable was a combination of

data frame columns, latitude and longitude, which we assigned to a variable “y”. We first did a correlation matrix to try and derive meaning from the features in the dataset, and a scatterplot (using package: Seaborn) to understand the layout of the coordinates.



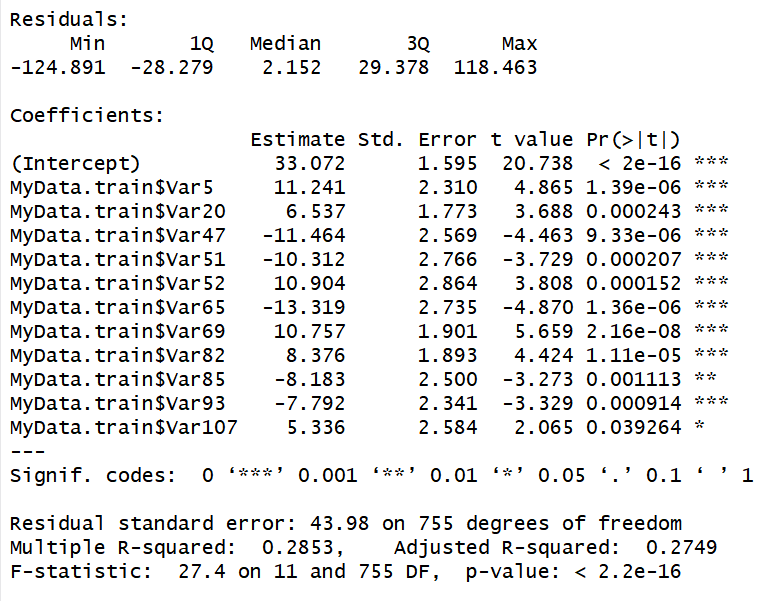
The correlation matrix was completely scattered, with many variables being highly positively and negatively correlated. Thus, it was very hard to find meaning from this without doing some sort of feature selection.



The coordinate map looked normal.

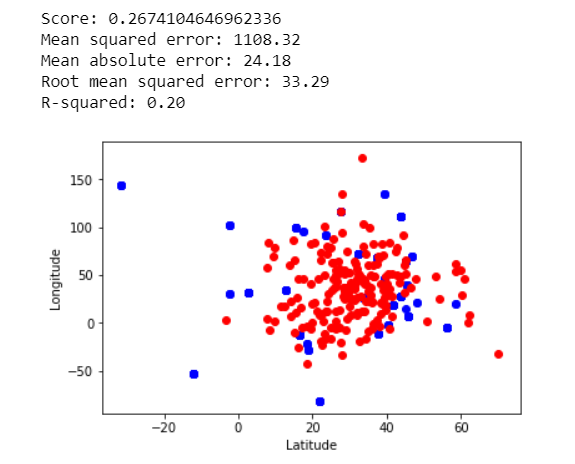
Due to the amount of highly correlated variables and severe need to trim down features, we first chose the simplest form of feature selection – forward selection. To do that, we used the Linear Regression and started by modeling ‘y’ on Var1 + Var2, then Var1 + Var2 + Var3, then Var1 + Var2 + Var3 + Var4, and so on, while removing insignificant features (having a p-value > .05) after each iteration. With that, we found the following subset of variables as the optimized set of features: ‘Var5’ + ‘Var20’ + ‘Var47’ + ‘Var51’ + ‘Var52’ + ‘Var65’ + ‘Var69’ + ‘Var82’ + ‘Var85’ + ‘Var93’ + ‘Var107’. That said, there were plenty of variables that were significant at one iteration, but not significant at others, meaning that there might be a better way to do this..

Here is a summary of the forward selection linear regression statistics:

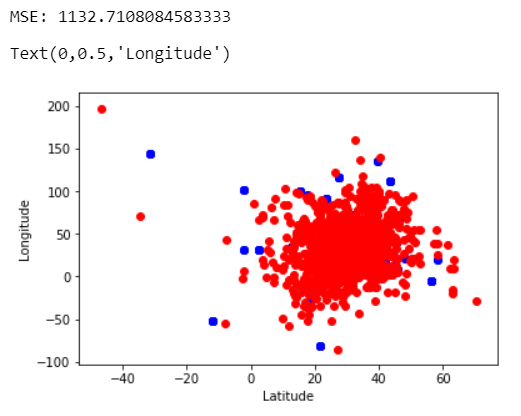


When trying to model only the ‘forward selected’ variables, the results were not good with an MSE > 1800 so we quickly decided to scrap that idea.

Next, using the Sklearn package with ‘LinearRegression.fit()’ and ‘LinearRegression.predict()’ in python, we fit the entire set of features to a linear model, and made predictions of Latitude and Longitude. Below is a plot of the predicted points (red) vs. the actual points (blue) as well as the score, mean squared error, mean absolute error, root mean squared error, and R-squared value:



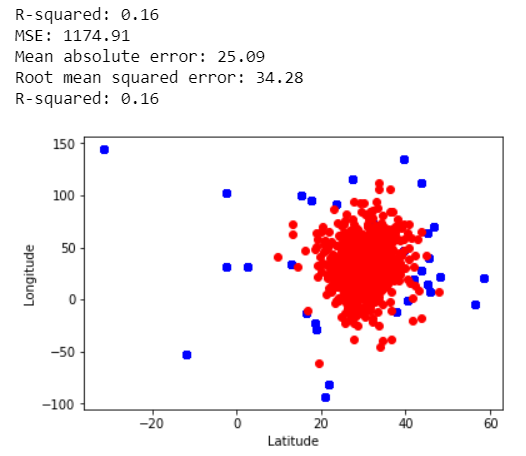
Next, we used ‘sklearn.model\_selection.cross\_val\_score’ in python with a cv=100 to cross validate this model. We tested many CV levels and found 100 to be optimal. The cross validated Linear Regression resulted in a higher MSE of 1132.71. Also, the predictions were noticeably more clustered around the center of the data which means the cross validating is working to make the predictions more mean-centric – however it could be at the expense of total MSE:

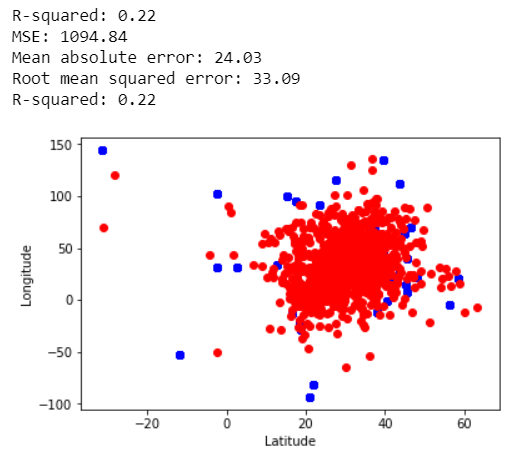


We were not satisfied with our results using this model, so we moved onto the next one.

**Model 2 & 3 - Lasso and Ridge**

Our motivation behind using Lasso and Ridge was subset selection. Again, since this dataset has few observations and a high count of features, it was unlikely that would be able to accurately predict a response using all the features. Ridge and Lasso introduces a tuning parameter, , that incurs a shrinkage penalty on the Beta’s of a model. Although they both incur a shrinkage penalty on X (features), Lasso completely removes unneeded variables from the model. Ridge will just highly penalize these features, making them more relevant to the model, but ultimately keeping them. In our Lasso model, we found, and CV = 7 to be optimal, and in Ridge we had and CV = 5 as being optimal. Here are our results for Lasso and Ridge, respectively:





We can see that Ridge performed better than Linear Regression and Lasso regression. This makes sense to us because of the predicament we descried earlier – many features being

significant and insignificant depending on the subset of features selected in the model. Lasso removed variables that were important and ultimately hurt the final model, while Ridge penalized them in an efficient way that optimized our model. Although we made progress, we were still not satisfied. The next model we moved to was Random Forest.

**Model 4 – Random Forest:**

Our motivation behind using Random Forest was also subset selection – we thought this could perform even better than Ridge. Random Forest is a form of Decision Tree classifier that partitions the feature space of the dataset, X, into non-overlapping regions. The difference with random forest is that is uses its own special form of “bagging”, which is a method of

bootstrapping, that decorrelates the bagged models and further reduce the model’s variance. For each bagged set – a decision tree is fit, but only uses a fraction of the variables for each split.

We setup up a Random Forest Model using K-folds cross validation as well as “MultiOutputRegressor” in Python. After setting up a mechanism to continuously output model results per distinct levels of parameters – specifically **‘max\_features’** and **‘n\_estimators’**, we were able to optimally tune the model. **‘Max\_features’** represent the number of features to consider when looking for the best split, and **‘n\_estimators’** specifies the number of trees in the forest. We used **‘max\_depth’** as a ‘conservative’ parameter to include, which represents the maximum depth of the tree. After much testing – we found the best fit parameters to be:

* **‘n\_estimators’** = 400
* **‘max\_features’** = 60,
* **‘max\_depth’** = 13.

**Conclusions:**

We conclude that for data science problems with a high feature space and low levels of observations, Random Forest is a good method to use. It solves this problem in NP-hard fashion fitting thousands if not more combinations of parameters to optimally find the best ‘Random Forest’ for this model. Although Random Forest worked best for us, each data science problem is unique in nature, so Random Forest may not always be the best choice.

Ridge regression was much better than Linear Regression when doing this problem. It was a more effective way to deal with high dimensionality. Linear regression cannot effectively fit the data without a penalty factor, which Ridge fixes. Lasso Regression seems to have drove

too many variables to zero resulting in a higher MSE than Ridge model. Overall, we were very impressed Random Forest and will use it when needed in the future. This project properly trained us to hit the ground running in our forthcoming profession – and we thank you for that!