Dumitrescu Gabriel Horia 333
Homework 1 – Dog Breeds

1) Data Analysis and pre-processing

a)Missing values

The height column contains missing values., so I replaced them with the mean of the column, instead of excluding their rows.

b)String Encoding

Since the labels are of type string I encoded them with a simple number. I encoded the features of type string with a One-Hot encoder, but I dropped the last column in order to avoid creating multicollinearity (also known as the Dummy Variable Trap).

c)Ratio

The data is pretty evenly distributed among the labels, with a ratio of 0.234 0.241 0.27 0.255. I tried balancing this ratio even further with an oversampling technique called SMOTE from imbalanced-learn (needed to run my program: https://pypi.org/project/imbalanced-learn/). The effect it had on performance was negligible.

d)Normalization

The numerical values in the data differed in scale by a large margin. I experimented with different scaling algorithms, until I found the best results. I standardized the training set for logistic regression, linear regression and knn regression and used a minmax scaler for knn and random forests.

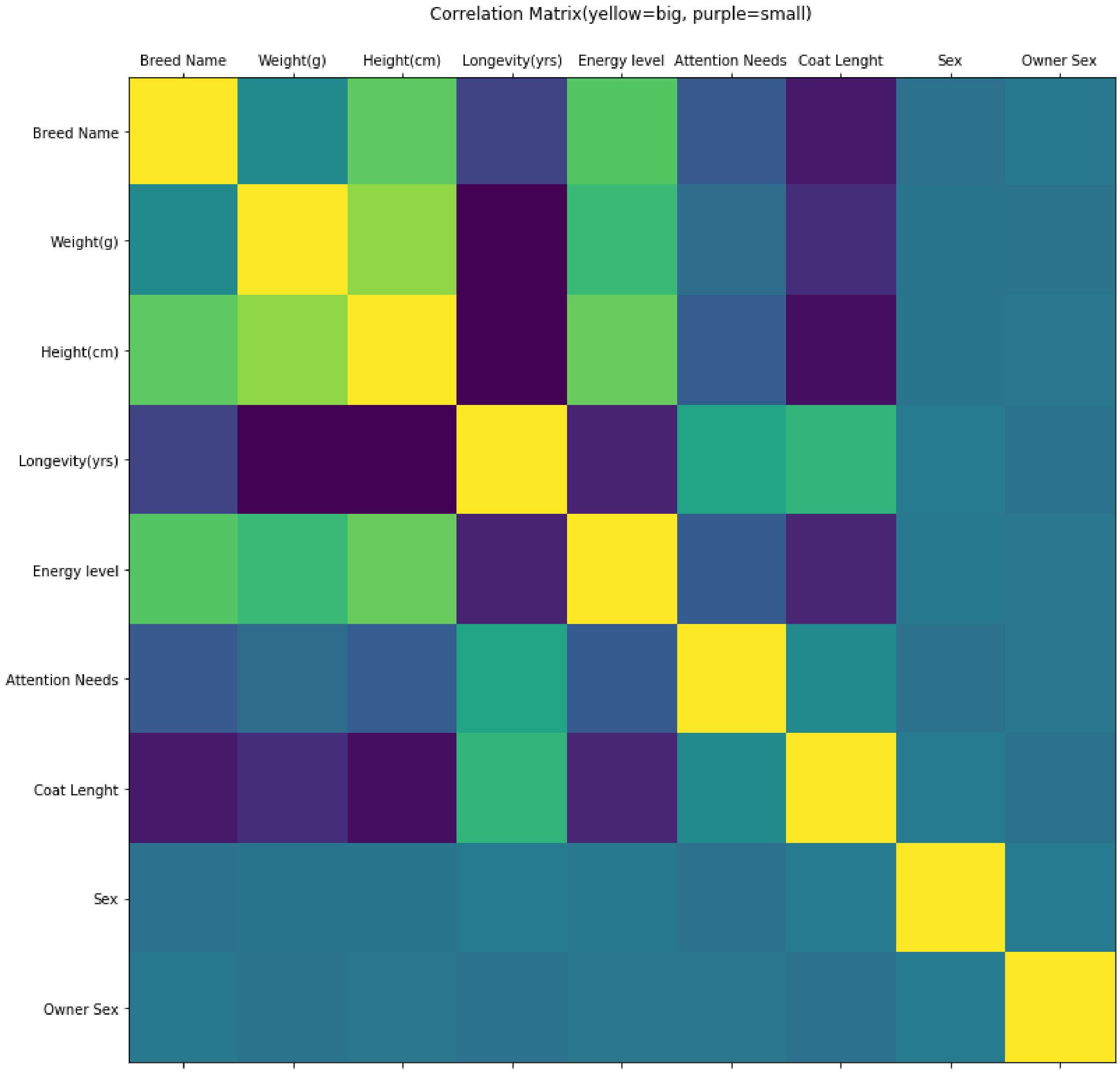
e)Feature engineering

I created a binary feature representing the sex of the owner. I computed this from the owner name with a function found on the internet (needed to run my program: https://pypi.org/project/gender-guesser/). It improved accuracy by a small margin.

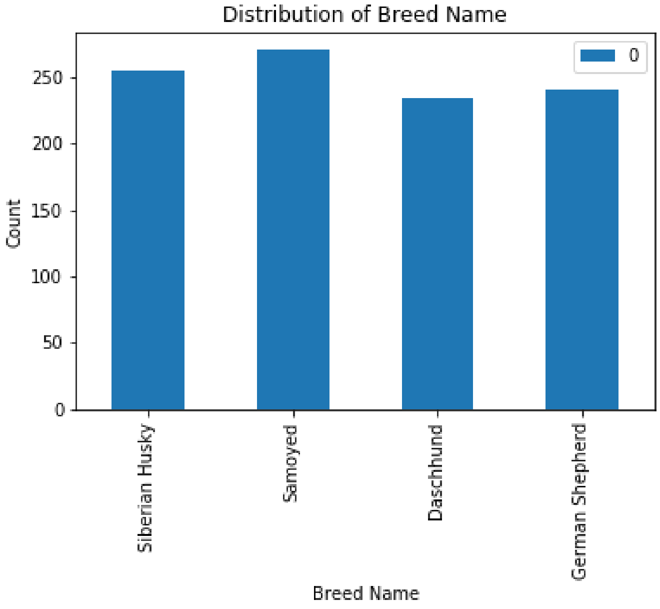
I experimented with creating another feature called “Big House Pet” to aid the distinction between Samoyeds and Siberian Huskies, since my classifiers mostly mistake there. It was 1 for the rows with a height around the average of Samoyed, a weight big enough to exclude Dachshunds and a medium energy level (typical of Huskies in this range) and 0 for the rest. This feature was useless.

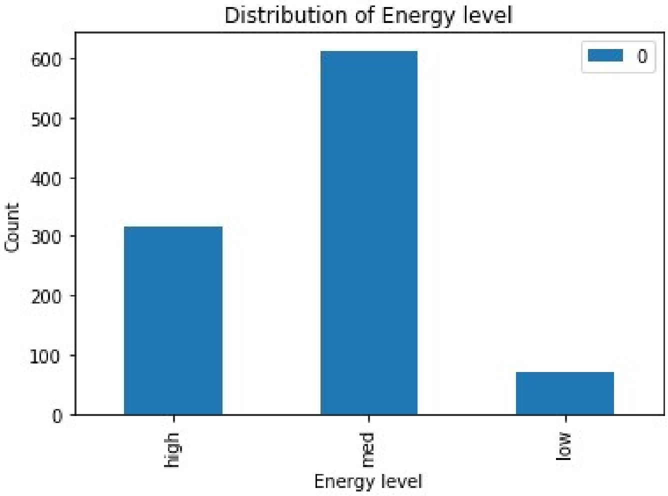
f) Feature Removal

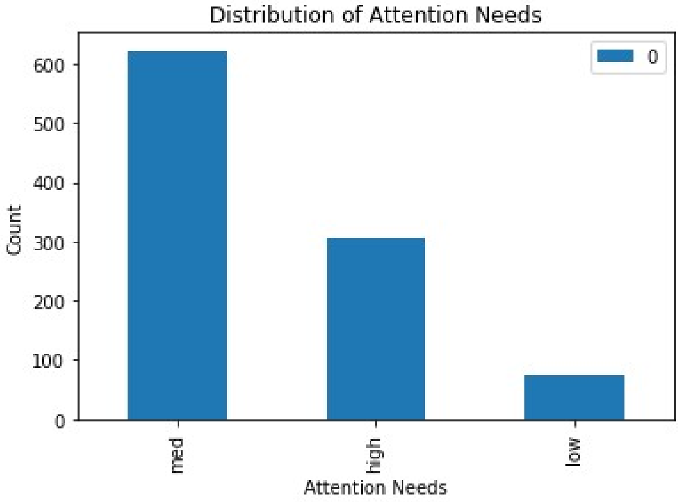
From the very start I excluded the owner name because it was affecting my accuracy. Later, I experimented with excluding other features that correlate poorly with breed names and longevity, but in the end, everything else was meaningful.

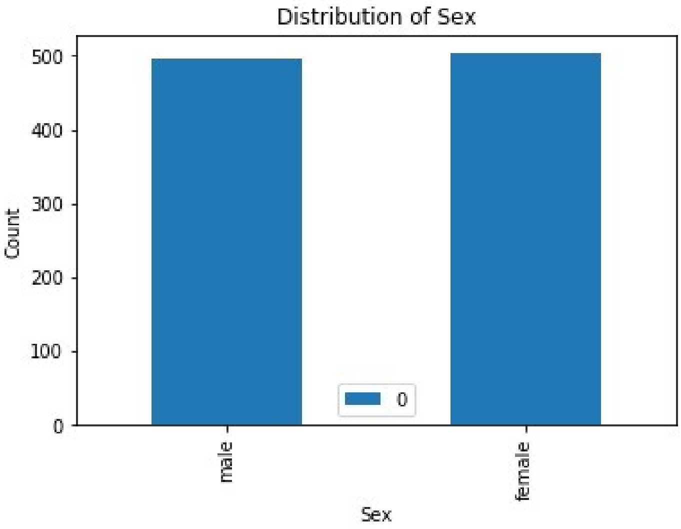


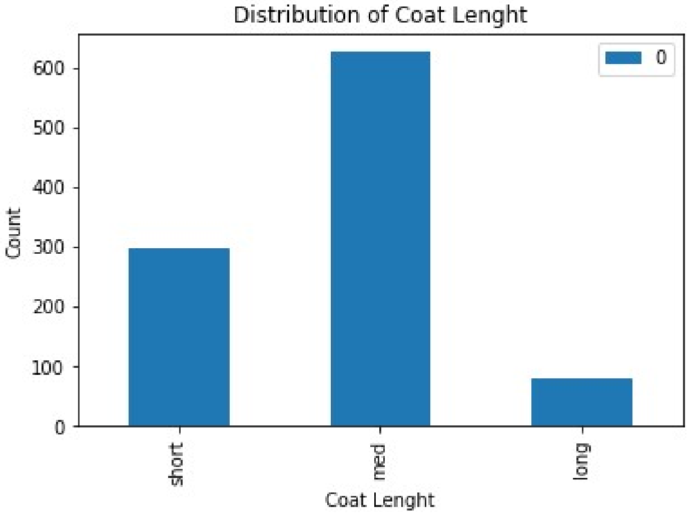
g)Distribution of features

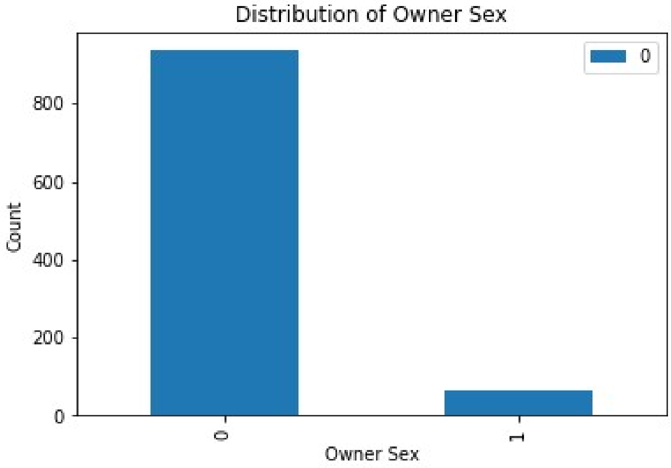






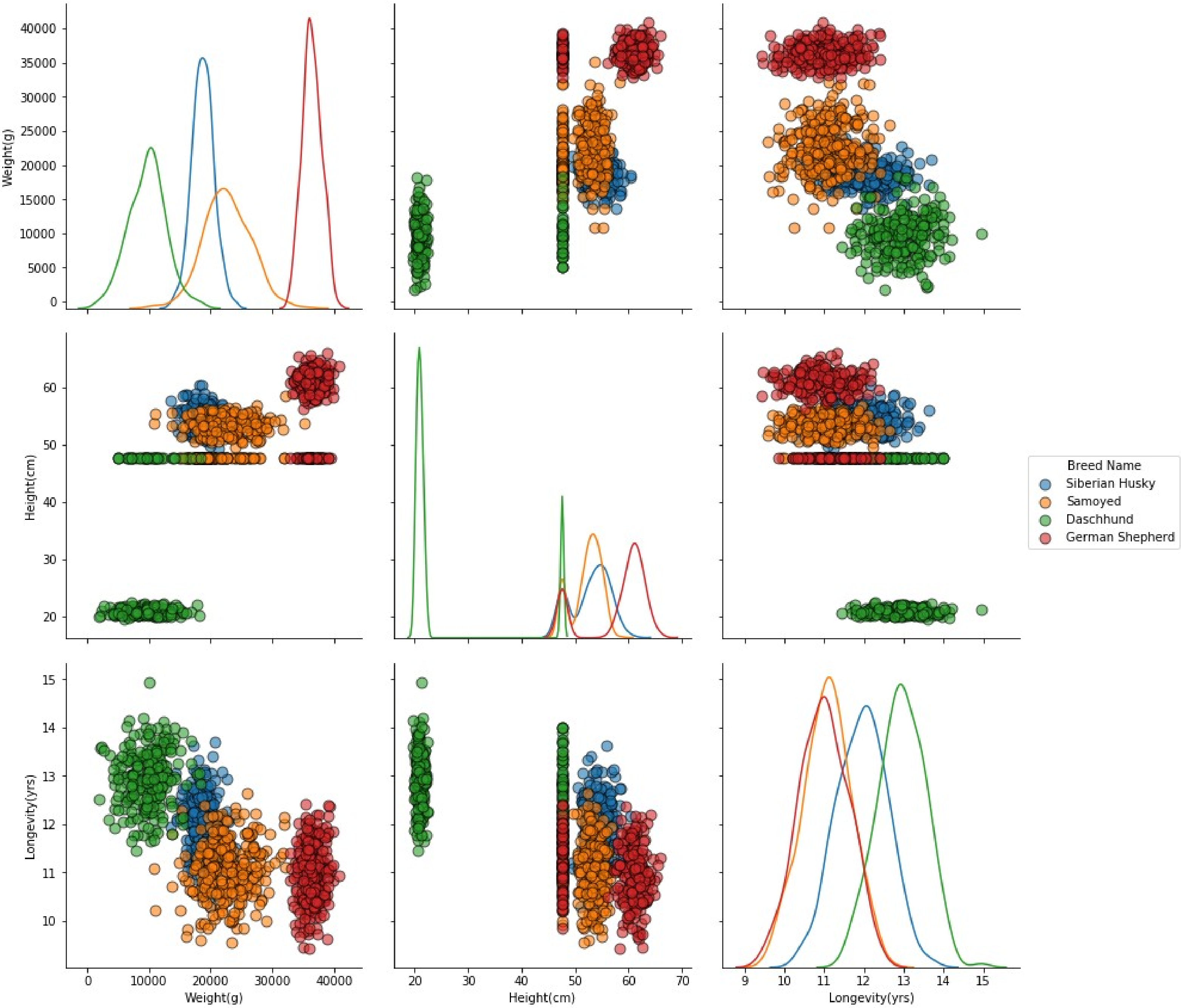

Distribution of Sex
500
400
300
200
100
0
Count
0
male
female
Sex


Distribution of Coat Lenght
600
500
400
300
200
100
0
Count
0
short
med
long
Coat Lenght


Distribution of Owner Sex
800
600
400
200
0
Count
0
0
1
Owner Sex

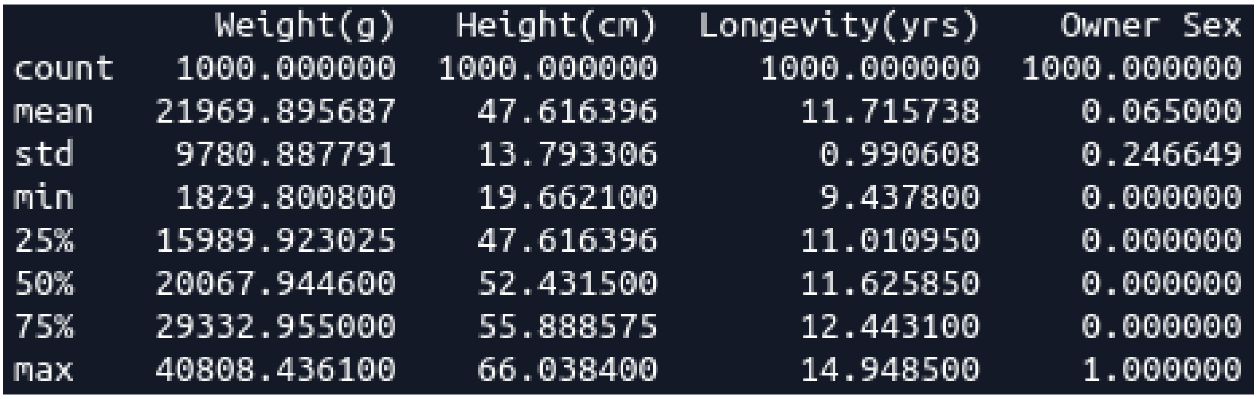
h)Data visualization

I used the seaborn library to visualize my data. I will show a smaller plot here, but I included the full plot in my archive.



i)Data statistics and understanding

To better understand my data I looked at statistics, I played around with different commands, looked trough my code with the debugger and checked Variable Explorer. I included below the output of a describe() command.



```
          Weight(g)   Height(cm)  Longevity(yrs)   Owner Sex
count   1000.000000  1000.000000     1000.000000 1000.000000
mean   21969.895687    47.616396       11.715738    0.065000
std     9780.887791    13.793306        0.990608    0.246649
min     1829.800800    19.662100        9.437800    0.000000
25%    15989.923025    47.616396       11.010950    0.000000
50%    20067.944600    52.431500       11.625850    0.000000
75%    29332.955000    55.888575       12.443100    0.000000
max    40808.436100    66.038400       14.948500    1.000000
```

j)Splitting

I liked the ratio of 0.75-0.25 best for my train-test sets. In the end, I used the Stratified ShuffleSplit cross-validator from sklearn to preserve the percentage of samples for each class. I didn’t split my training set further into validation and training because random search does it itself.

2) Classification

a)Logistic Regression

The basic logistic regression from sklearn had a decent performance for my first attempt: 84% accuracy. After hand-tuning it, my results improved even more.

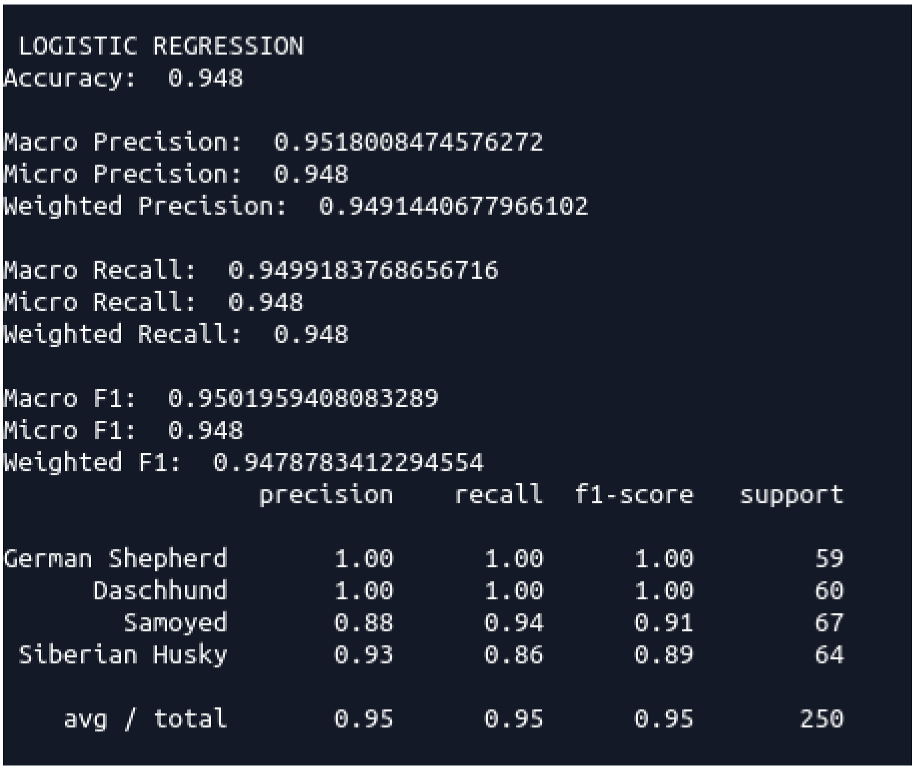
I tried making my regression polynomial by using PolynomialFeatures from sklearn, but it didn’t help too much.

In my hyper-parameter tuning quest I found the technique of random search, so I left it running overnight with as many hyper-parameters as I could imagine and stored the best results in a local file. After hand-tuning those I had something like 94% accuracy.

While researching ways to improve my data, I found a technique called Bootstrap aggregating, which basically creates a random forest of logistic regressions. It yielded even better results when combined with my optimal logistic regression. I left a random search running for 40 minutes to tune the parameters of that as well.

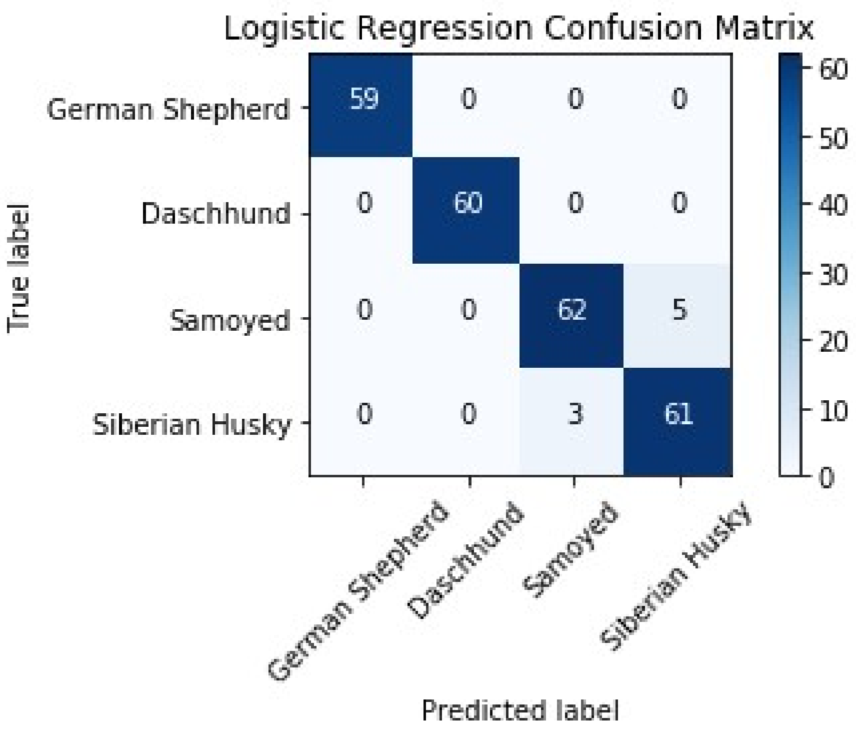
My accuracy improved when I found out about the stratified split as well. The maximal consistent result was 97.2% accuracy, but with program changes it fell to 95%. I evaluated many other meaningful scores and I think, in this situation, accuracy proved to be representative for model performance.

I displayed here my model general performance.

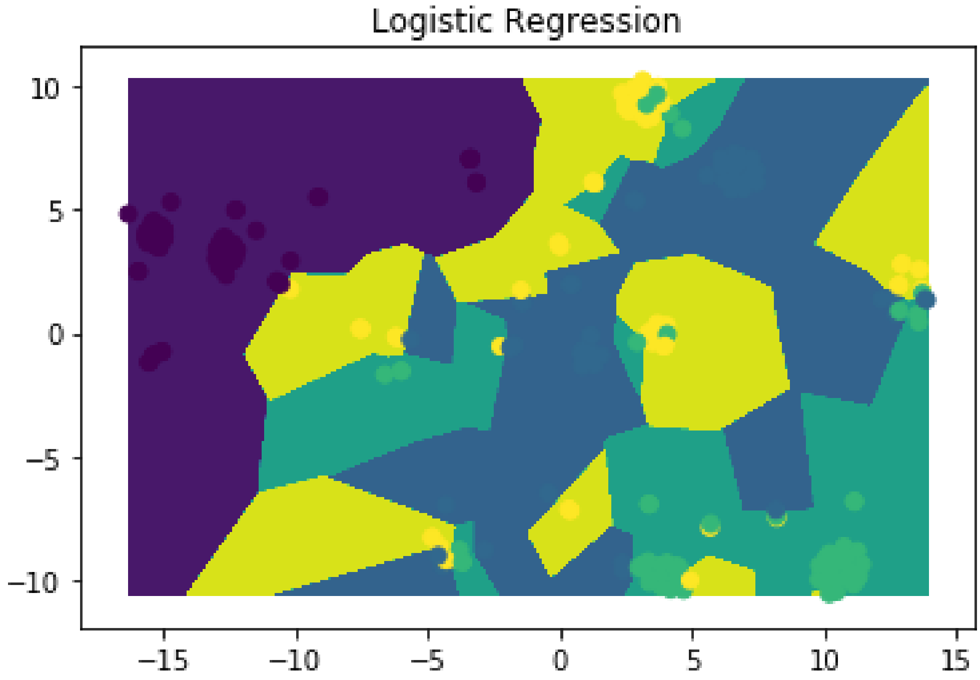


```
 LOGISTIC REGRESSION
Accuracy:  0.948

Macro Precision:  0.9518008474576272
Micro Precision:  0.948
Weighted Precision:  0.9491440677966102

Macro Recall:  0.9499183768656716
Micro Recall:  0.948
Weighted Recall:  0.948

Macro F1:  0.9501959408083289
Micro F1:  0.948
Weighted F1:  0.9478783412294554
                 precision    recall  f1-score   support

German Shepherd       1.00      1.00      1.00        59
      Daschhund       1.00      1.00      1.00        60
        Samoyed       0.88      0.94      0.91        67
 Siberian Husky       0.93      0.86      0.89        64

    avg / total       0.95      0.95      0.95       250
```

I also computed a confusion matrix which helped me find out my model confuses Samoyeds and Siberian Huskies. All of the following models have this problem. This is when I tried adding a feature to help discriminate among the two, but was unsuccesful.



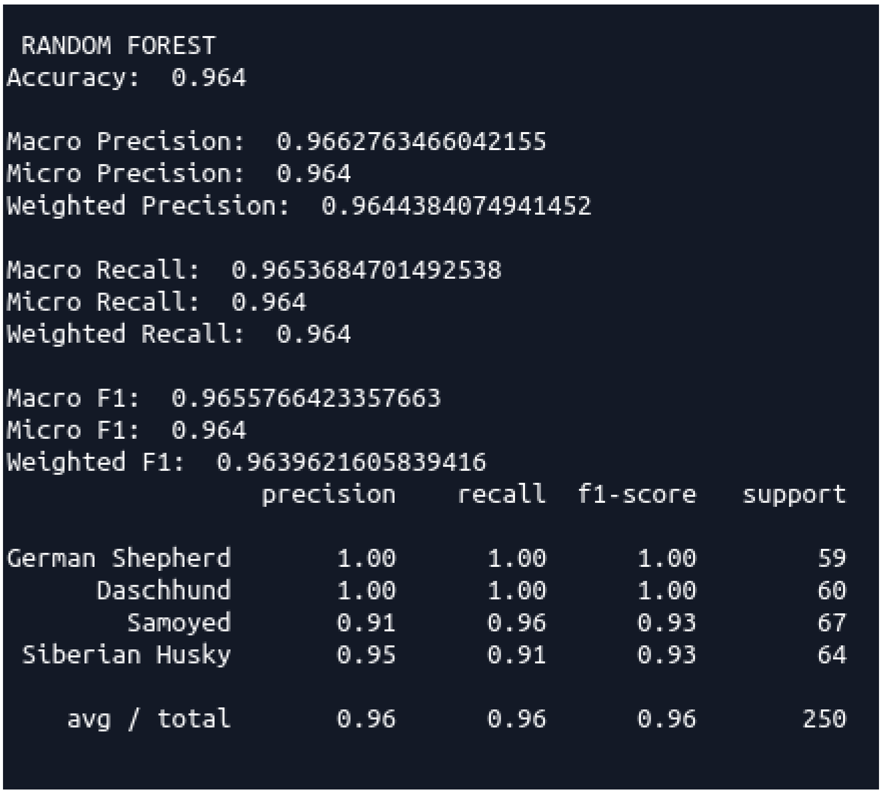
I also tried plotting the decision boundary of my model with an algorithm that combines the t-SNE visualization with vornoi diagrams, but I think the final result wasn't very representative. With my current knowledge, I would have tried using PCAs for this task.



b)Random Forest

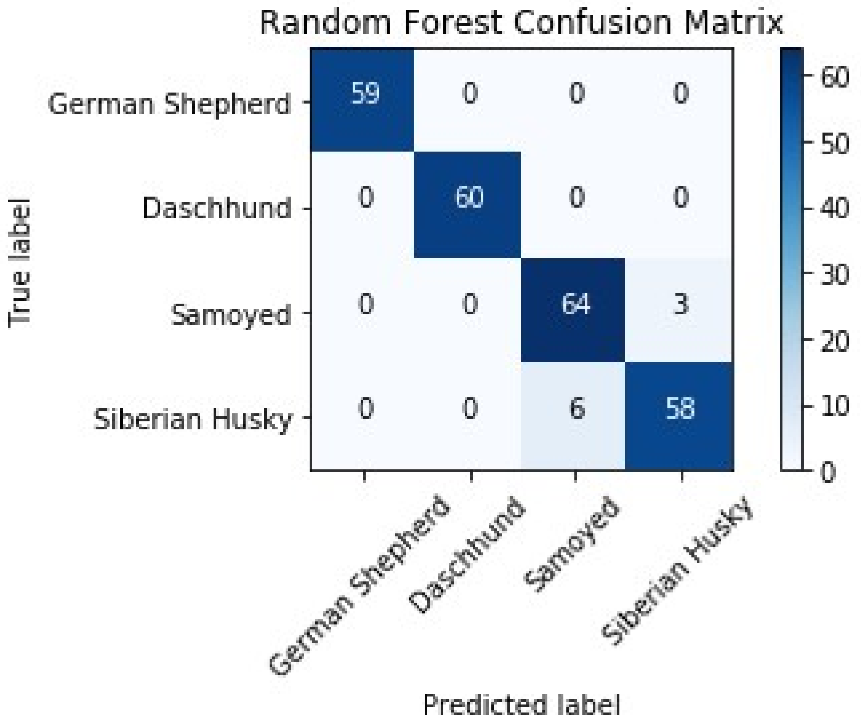
Even though, in the beginning it looked like logistic regression was going to be the best model for this task, in the end, random forests have the best performance. I tuned them after running my random search overnight. I used the model directly from sklearn.

It yielded good performance on all scores.

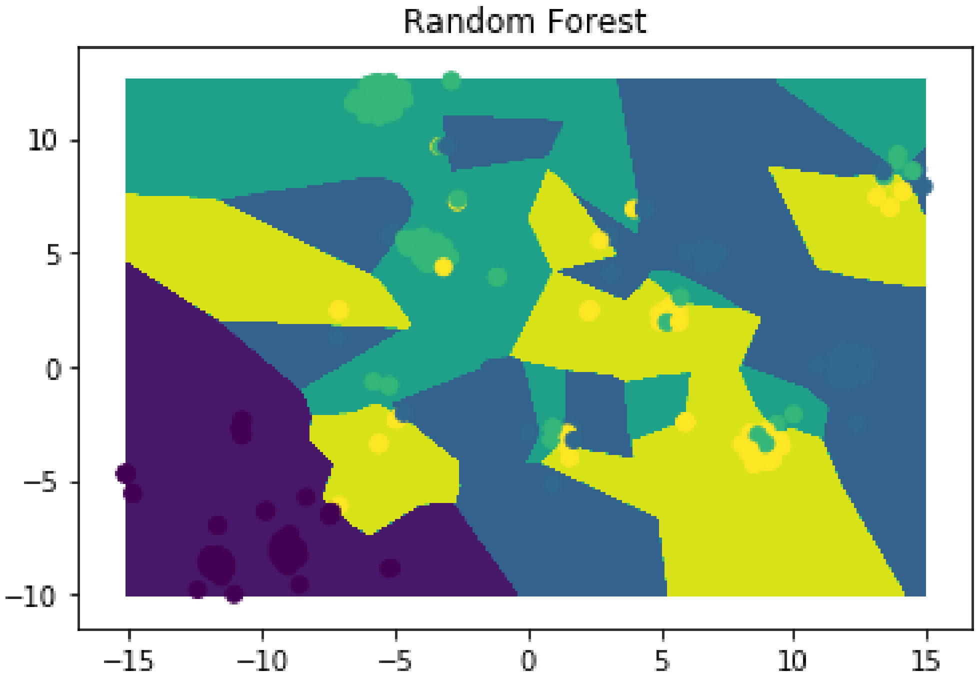


```
 RANDOM FOREST
Accuracy:  0.964

Macro Precision:  0.9662763466042155
Micro Precision:  0.964
Weighted Precision:  0.9644384074941452

Macro Recall:  0.9653684701492538
Micro Recall:  0.964
Weighted Recall:  0.964

Macro F1:  0.9655766423357663
Micro F1:  0.964
Weighted F1:  0.9639621605839416
                 precision    recall  f1-score   support

German Shepherd       1.00      1.00      1.00        59
      Daschhund       1.00      1.00      1.00        60
        Samoyed       0.91      0.96      0.93        67
 Siberian Husky       0.95      0.91      0.93        64

    avg / total       0.96      0.96      0.96       250
```

But they suffer of the same flaw as all of my classification models, they confuse Samoyeds with Siberian Huskies. I’m really curious if there is an obvious way that I’m missing to help the model discriminate in this specific situation. If there is, please, can you send me an email?



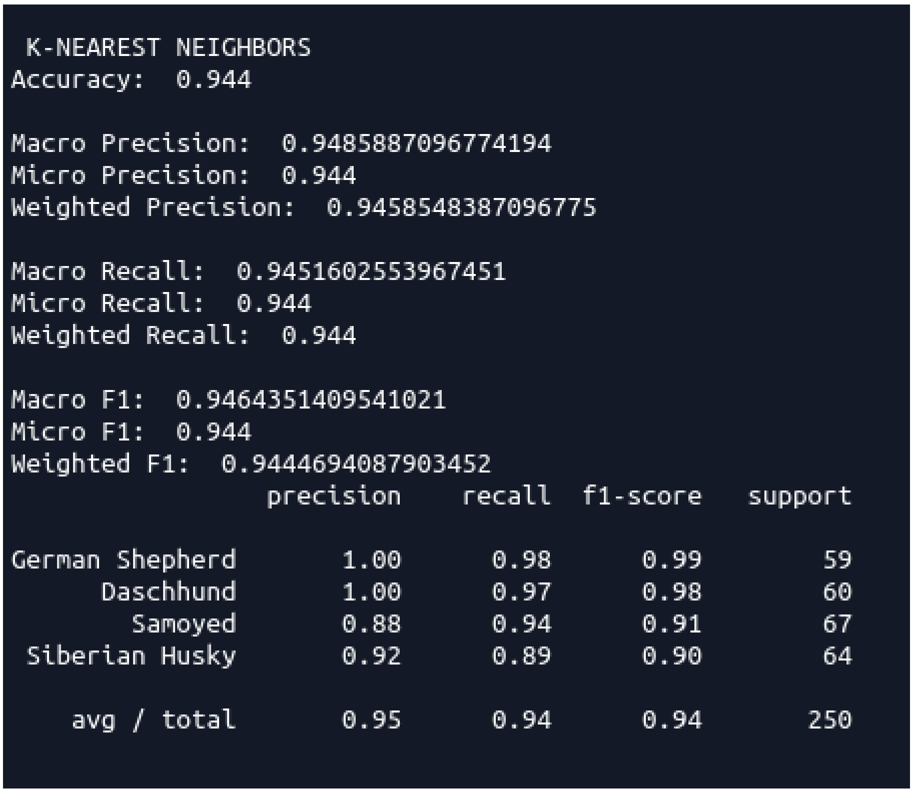
And this is the decision boundary.



c)*k*-nearest neighbors algorithm

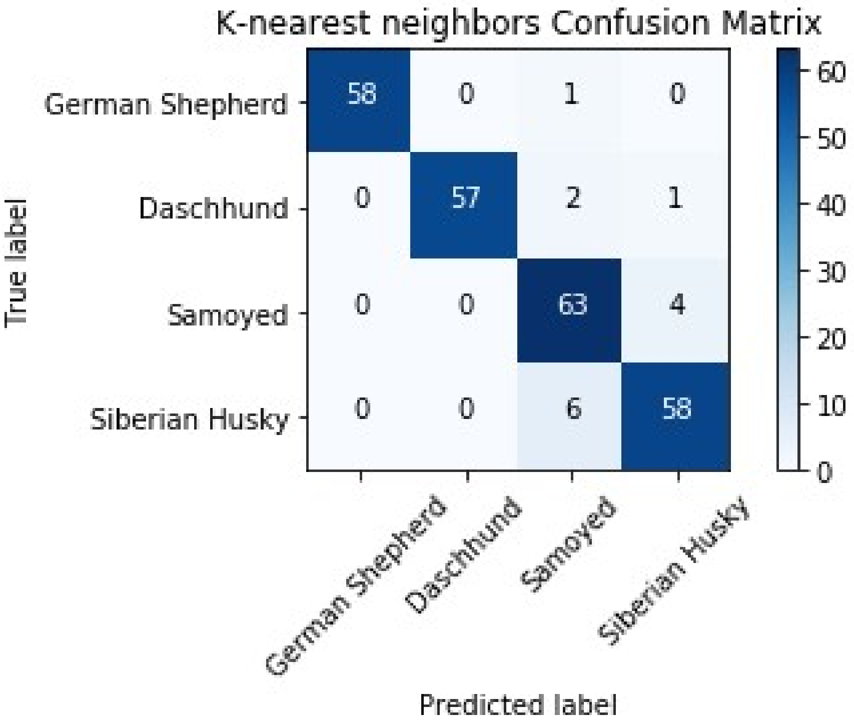
For knn, I used a similar approach to that of the logistic regression. I found my bets hyper-parameters with random search. And with this model, I searched for the best hyper-parameters for the BaggingClassifier. In the beginning it had terrible performance, but it progressively climbed to 94.4%. It’s still my weakest model.

The same as with the other classification models, accuracy is representative of performance in this case.

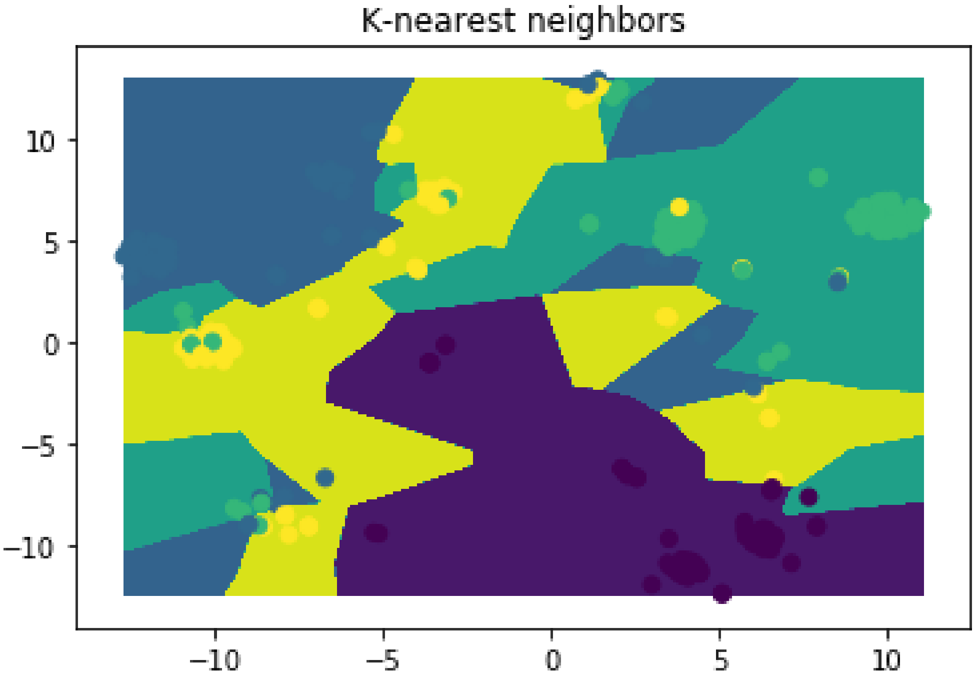


```
 K-NEAREST NEIGHBORS
Accuracy:  0.944

Macro Precision:  0.9485887096774194
Micro Precision:  0.944
Weighted Precision:  0.9458548387096775

Macro Recall:  0.9451602553967451
Micro Recall:  0.944
Weighted Recall:  0.944

Macro F1:  0.9464351409541021
Micro F1:  0.944
Weighted F1:  0.9444694087903452
                 precision    recall  f1-score   support

German Shepherd       1.00      0.98      0.99        59
      Daschhund       1.00      0.97      0.98        60
        Samoyed       0.88      0.94      0.91        67
 Siberian Husky       0.92      0.89      0.90        64

    avg / total       0.95      0.94      0.94       250
```

This mistakes more classes than just Samoyeds and Siberian Huskies. I’m guessing this is where the performance difference between this and the other 2 models lies.



This decision boundary visualization should be the closest to reality out of the 3, because it’s approximated by another knn model.



3) Regression

a)Linear Regression

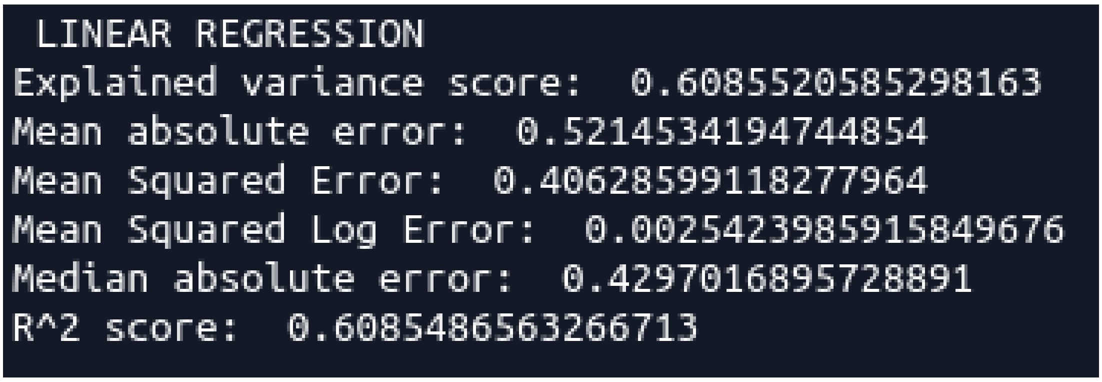
For this, I used the linear_model package from sklearn. I played around with a simple linear regression, ridge regression, lasso regression and elastic net. They all yielded a very similar performance around 0.6. After tunning them with a random search, some improved a tiny bit.

I found an analog of the Bagging Classifier for regression, the Bagging Regressor, but sadly, even after tuning it with a random search, it lowered my score.

I even experimented with normalizing the labels, just in case I could improve this further.

Using PolynomialFeatures to make my regression polynomial improved my score by a very small margin.

My final R^2 score is 0.61. And all approaches score similarly. In the end, I like simple linear regression with polynomial features the most because of it’s simplicity.



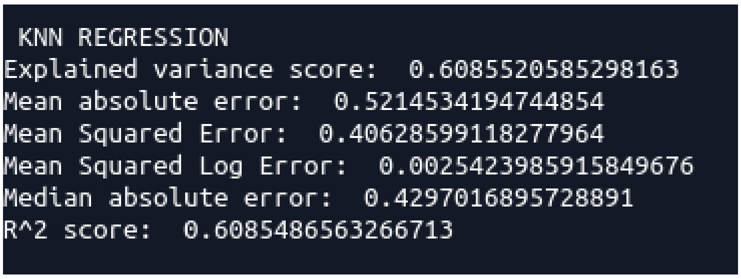
```
 LINEAR REGRESSION
Explained variance score:  0.6085520585298163
Mean absolute error:  0.5214534194744854
Mean Squared Error:  0.40628599118277964
Mean Squared Log Error:  0.0025423985915849676
Median absolute error:  0.4297016895728891
R^2 score:  0.6085486563266713
```

b)*k*-nearest neighbors regression

For knn regression, I also searched for the best hyper-parameters with a random search and hand-tuned them afterwards.

I tried the Bagging Regressor but it was decreasing my performance.

In the end, this is the better regression out of the 2. It still has around the same performance, an R^2 score of 0.61. The difference is it’s usually slightly bigger and more consistent.



```
 KNN REGRESSION
Explained variance score:  0.6085520585298163
Mean absolute error:  0.5214534194744854
Mean Squared Error:  0.40628599118277964
Mean Squared Log Error:  0.0025423985915849676
Median absolute error:  0.4297016895728891
R^2 score:  0.6085486563266713
```