Programming Assignment F:

The purpose of this programming assignment is to determine the top three features from Wine Dataset using random forests method. Furthermore, we will investigate the popular clustering methods (K-Means, Birch, Agglomerative Clustering, and DBSCAN) using those top three features that we determined from random forests method. We will run each clustering method for 1, 3, 5, 11 clusters, and provide a 3D graph visualization as well as runtime for each iteration. The programming assignment is done in Python 3.6 using sklearn package (http://scikit-learn.org/). The hardware includes a MacBook Pro with Intel core i7 and 16GB of RAM with an operating system of OSX Version 10.12.3.

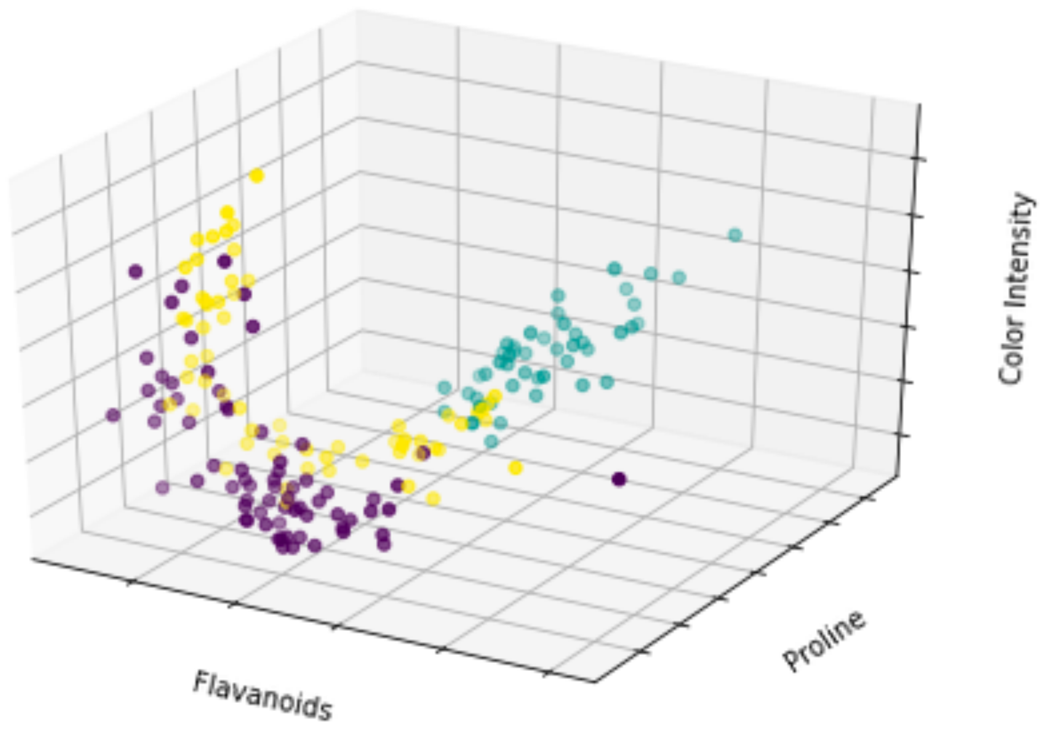
1) For the random forest method, I used RandomForestClassifier ensemble with n_estimators= 1000 and criterion = 'entropy' as parameters. The parameter, n_estimators, is the number of trees that will be generated in the algorithm. The higher number of trees increases the accuracy of determining the importance of the features. Random Forest essentially takes an average of the all values in the trees. In this case, I have given 1000 trees, which might have been more than enough, in order to accurately determine the top three features. Therefore, the algorithm returns all the features with their respective value below. The top 3 features are: **Flavanoids, Proline, Color Intensity.**

Flavanoids	0.17783608295298423
Proline	0.15316211932034413
Color Intensity	0.13859670997925955
OD280/OD315 of diluted wines	0.13543375734648969
Alcohol	0.11724624000008846
Hue	0.081493384905736613
Total phenols	0.05901134355487616
Malic Acid	0.030775215634327207
Magnesium	0.028231940198704893
Alcalinity of Ash	0.025352049995492284
Proanthocyanins	0.019412659155907064
Ash	0.013472501555337116
Nonflavanoid phenols	0.0099182386198699863

2) The first clustering method, K-Means, will take parameter n_clusters = 3 against the top 3 features as determined above. In order to generate the run time of K-Means fitting, I use another python library call timeit to output the running time. I start the timer before the fitting function and stop after the fitting function. Running the algorithm 5 different times:

Run 1	0.016372540994780138
Run 2	0.015201788017293438
Run 3	0.015945635008392856
Run 4	0.013141681993147358
Run 5	0.011502273991936818

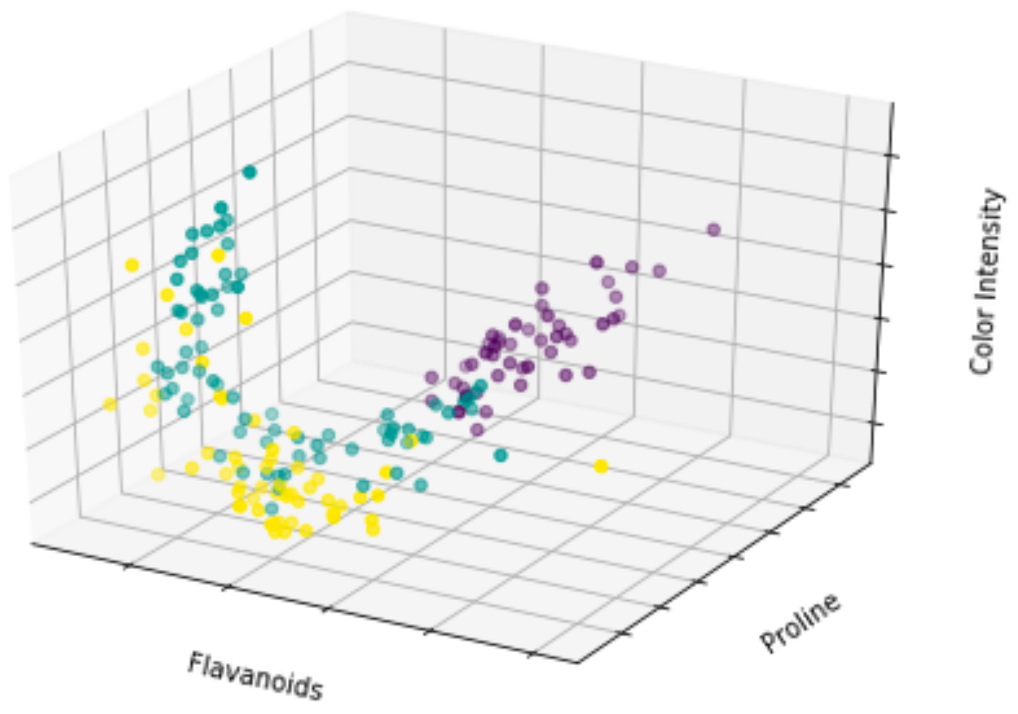
The average running time is about **14.42 milliseconds** to fit 3 clusters in K-means method. Below is the 3d graph visualization with 3 clusters.



The second clustering method, Birch, takes in parameter n_clusters = 3 against the top 3 features as determined above, branching factor default to 50 and threshold (diameter of sub-cluster) default to 0.5. I start the timer before the fitting function and stop after the fitting function. Running the algorithm 5 times:

Run 1	0. 010433707997435704
Run 2	0. 009979043010389432
Run 3	0. 008869687997503206
Run 4	0. 008747225016122684
Run 5	0. 009066089987754822

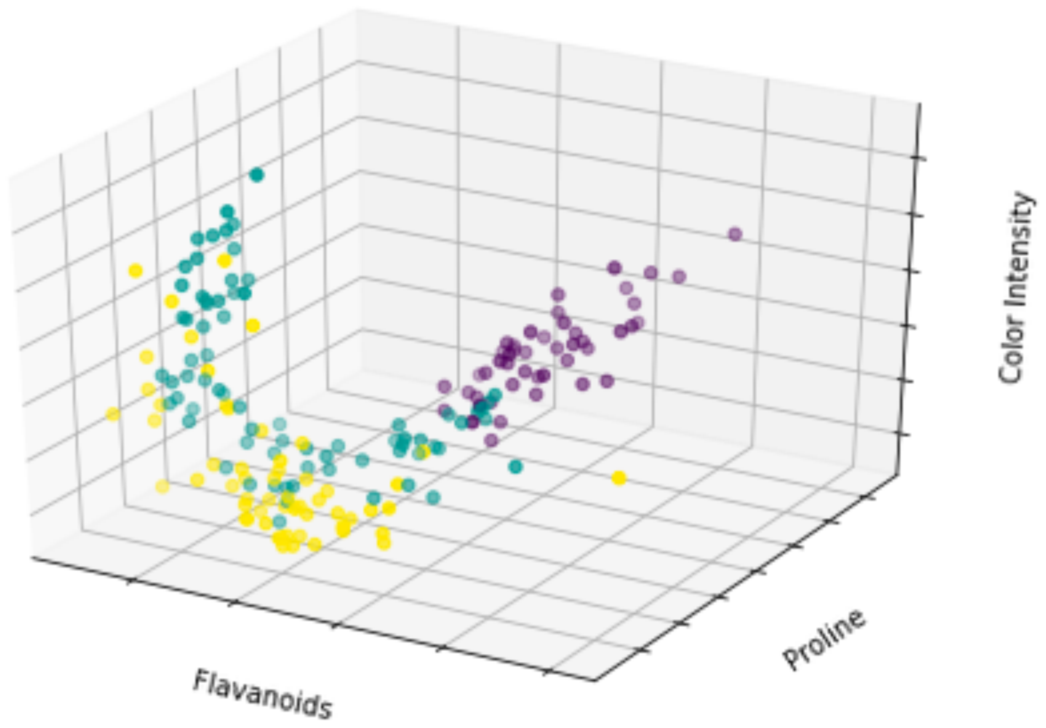
The average running time is about **9.42 milliseconds** to fit 3 clusters in Birch method. Below is the 3d graph visualization with 3 clusters.



The third clustering method, Agglomerative Clustering, will take parameter n_clusters = 3 against the top 3 features, affinity is set to default which is Euclidean, and linkage that is set to ward (minimizes the variance of the clusters being merged) by default. I start the timer before the fitting function and stop after the fitting function. Running the algorithm 5 times:

Run 1	0. 0011187860218342394
Run 2	0. 001062514988007024
Run 3	0. 001056721986969933
Run 4	0. 0011835710029117763
Run 5	0. 001036215981002897

The average running time is about **1.09 milliseconds** to fit 3 clusters in Agglomerative method. Below is the 3d graph visualization with 3 clusters.



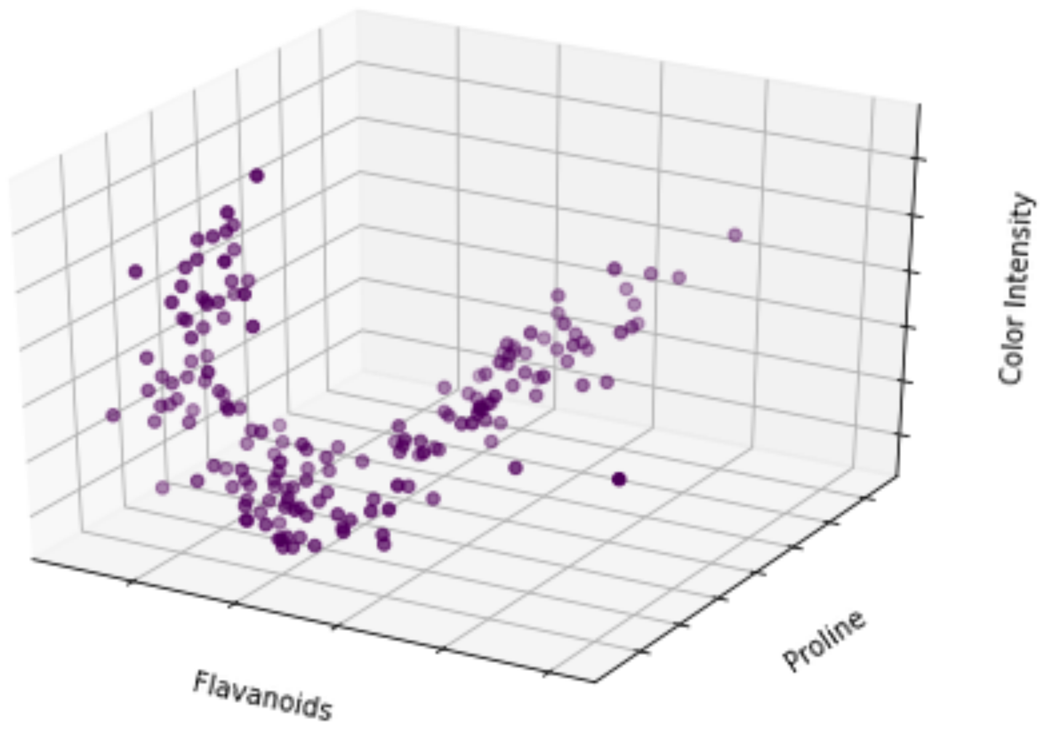
3) K-Means

Running K-means method against 1 cluster for the top 3 features as determined above. I start the timer before the fitting function and stop after the fitting function. Running the algorithm 5 times:

n clusters = 1

Run 1	0. 0036769699945580214
Run 2	0. 003603240998927504
Run 3	0. 003719937987625599
Run 4	0. 003697048989124596
Run 5	0. 004235491011058912

The average running time is about **3.78 milliseconds** to fit 1 cluster in K-Means method, which is faster than fitting 3 clusters. Intuitively, running K-means with 1 cluster is much faster because it does not require the algorithm to reexamine every point against the new centers and to recalculate the mean center points. Below is the 3d graph visualization with 1 clusters.

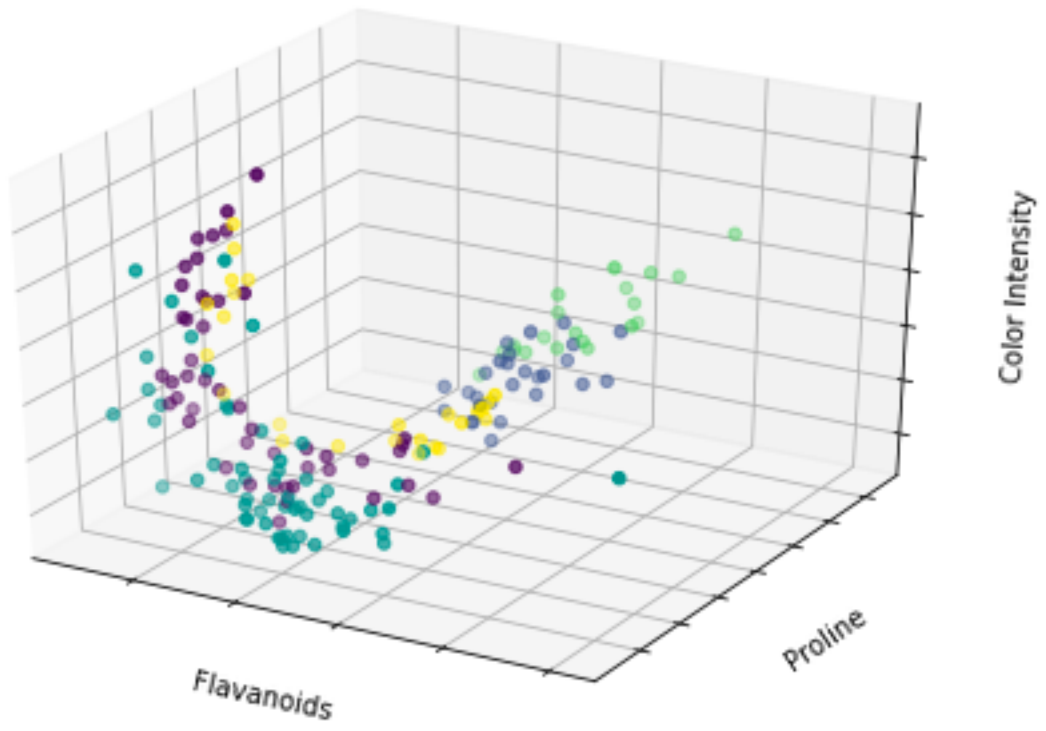


Running K-means method against 5 cluster for the top 3 features. I start the timer before the fitting function and stop after the fitting function. Running the algorithm 5 times:

n clusters = 5

Run 1	0. 015278347011189908
Run 2	0. 01614384798449464
Run 3	0. 014840518997516483
Run 4	0. 014069013006519526
Run 5	0. 018971159995999187

The average running time is about **15.864 milliseconds** to fit 5 clusters in K-Means method, which is slightly slower from what I observed for 3 clusters. Interestingly, running against 5 clusters should be slower than running 3 clusters, however, the average running time is comparable to running 3 clusters. One explanation could be that running K-Means against 5 clusters did not require the algorithm to recalculate the centers again, or had minor recalculation of the mean centers. I also noticed that K-means method was not able to handle non-convex clusters correctly. Below is the 3d graph visualization with 5 clusters.

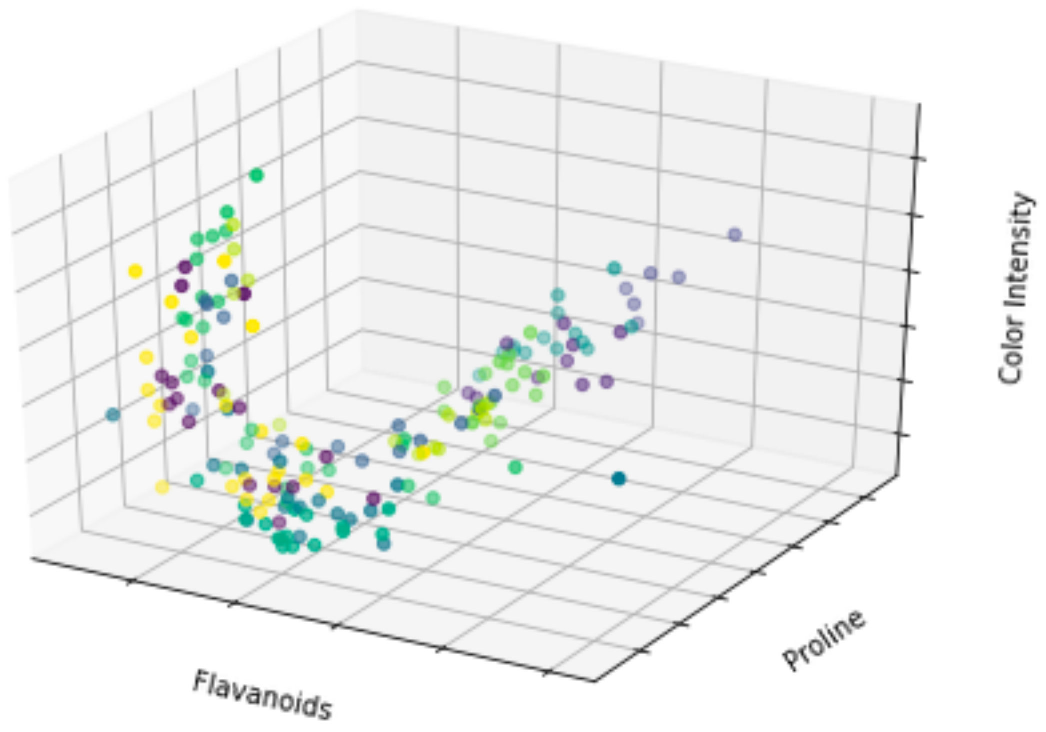


Running K-means method against 11 cluster for the top 3 features as determined above. I start the timer before the fitting function and stop after the fitting function. Running the algorithm 5 times:

n clusters = 11

Run 1	0. 017634788004215807
Run 2	0. 021763442986411974
Run 3	0. 021575298014795408
Run 4	0. 013702736992854625
Run 5	0. 02184795998618938

The average running time is about **19.3 milliseconds** to fit 11 clusters in K-Means method, which is significantly slower than 3 clusters as expected. Intuitively, this makes sense since the more clusters require more reexamining of mean points and more recalculating of the new means points. Below is the 3d graph visualization with 5 clusters.



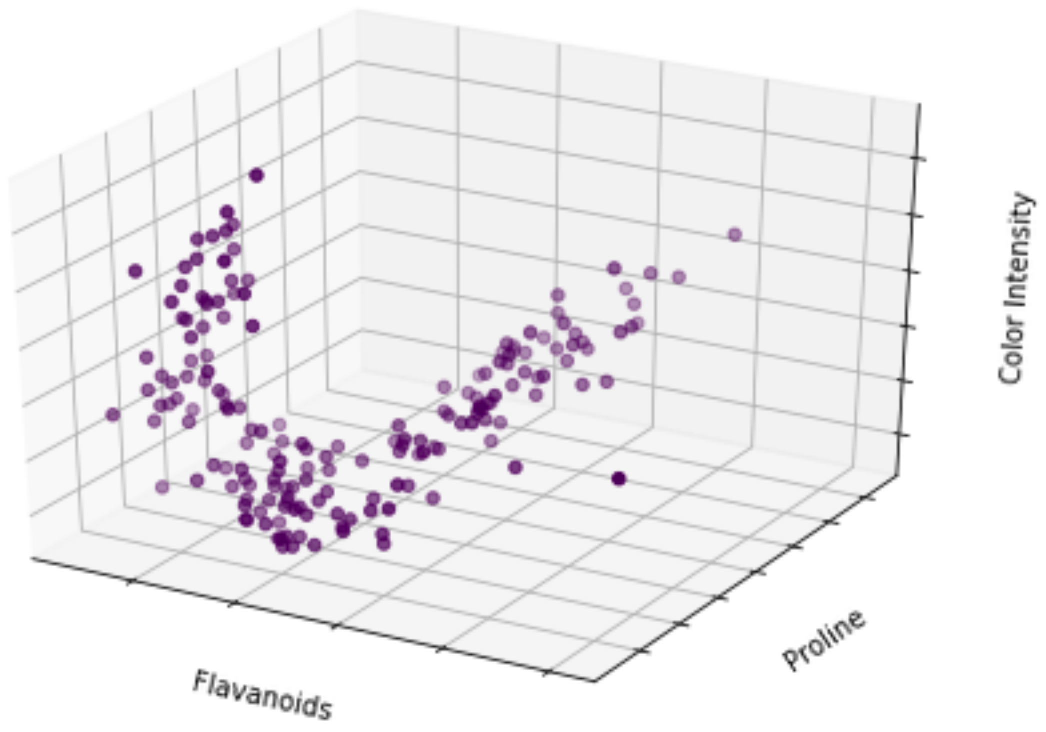
Birch

Running Birch method against 1 cluster for the top 3 features as determined above. I start the timer before the fitting function and stop after the fitting function. Running the algorithm 5 times:

n clusters = 1

Run 1	0. 006875534018035978
Run 2	0. 0070446610043291
Run 3	0. 006853063008747995
Run 4	0. 006931835989234969
Run 5	0. 007208497991086915

The average running time is about **6.98 milliseconds** to fit 1 cluster which is faster than 3 cluster. Below is the 3d graph visualization with 1 clusters.

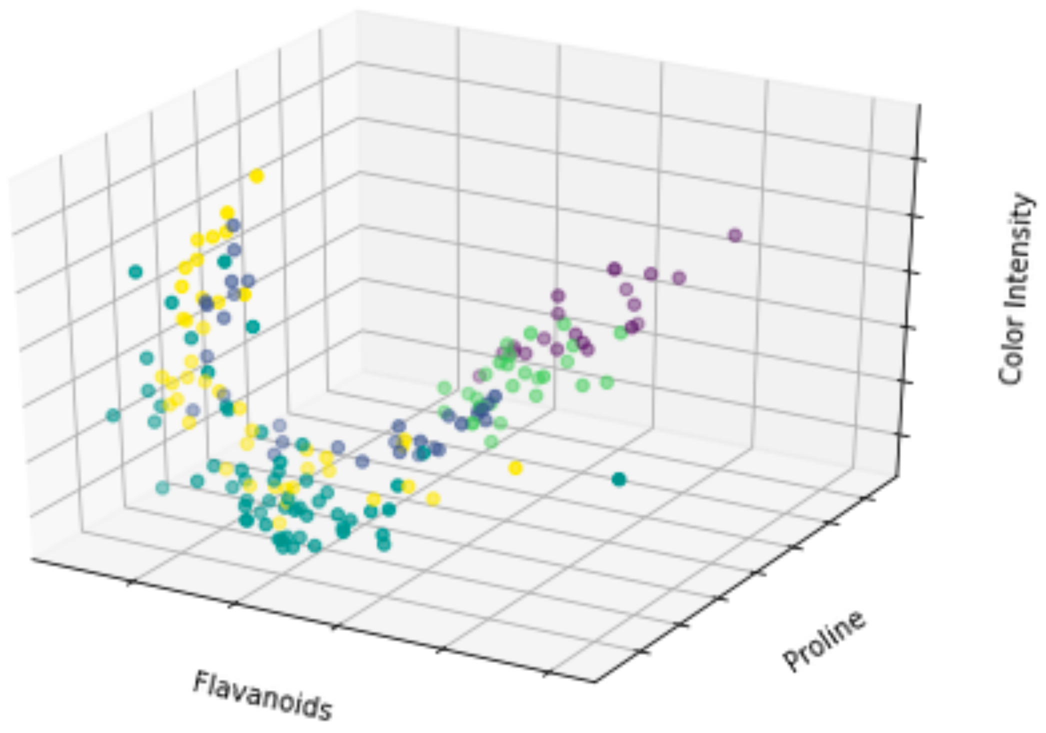


Running Birch method against 5 clusters for the top 3 features as determined above. I start the timer before the fitting function and stop after the fitting function. Running the algorithm 5 times:

n clusters = 5

Run 1	0. 007144284987589344
Run 2	0. 006923697976162657
Run 3	0. 006931835989234969
Run 4	0. 006867500982480124
Run 5	0. 007241435989271849

The average running time is about **5.65 milliseconds** to fit 5 cluster. Interestingly, comparing the running time in this iteration to running 1 cluster iteration above, the difference in running time is minimal. This makes sense because Birch method only scans the data once and does not need to reexamine each point again.

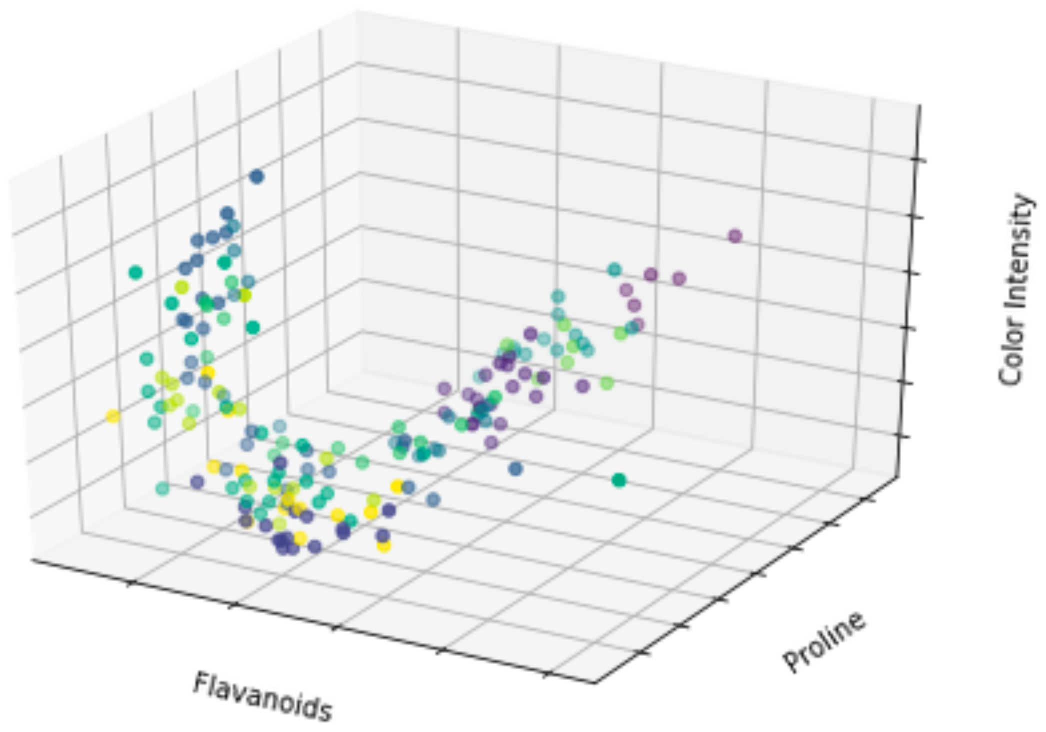


Running Birch method against 11 clusters for the top 3 features as determined above. I start the timer before the fitting function and stop after the fitting function. Running the algorithm 5 times:

n clusters = 11

Run 1	0. 007426388008752838
Run 2	0. 008581353002227843
Run 3	0. 009152141021331772
Run 4	0. 008396883000386879
Run 5	0. 008861561014782637

The average running time is about **8.48 milliseconds** to fit 5 cluster. The running time for 11 clusters is slightly slower than running time for 5 clusters, however, the difference is still minimal. As I discussed from previous iteration, BIRCH only requires a single scan of data to build the CF tree in memory. Comparing to K-Means, Birch is much faster and efficient because it avoids re-examining all the points.



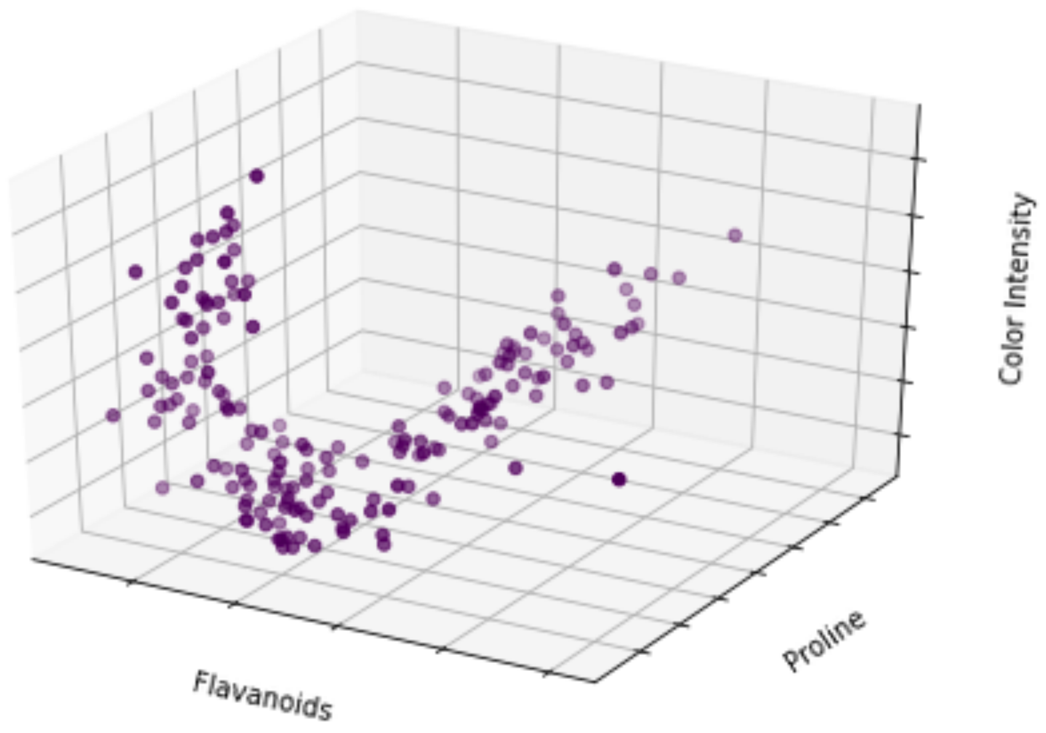
Agglomerative Clustering

Running Agglomerative Clustering method against 1 clusters for the top 3 features as determined above. I start the timer before the fitting function and stop after the fitting function. Running the algorithm 5 times:

n clusters = 1

Run 1	0. 0007832249975763261
Run 2	0. 000833658006740734
Run 3	0. 0008580129942856729
Run 4	0. 0008462440164294094
Run 5	0. 0008289699908345938

The average running time is about **8.48 milliseconds** to fit 1 cluster. Comparing to the same method running at 3 clusters, it is very comparable.

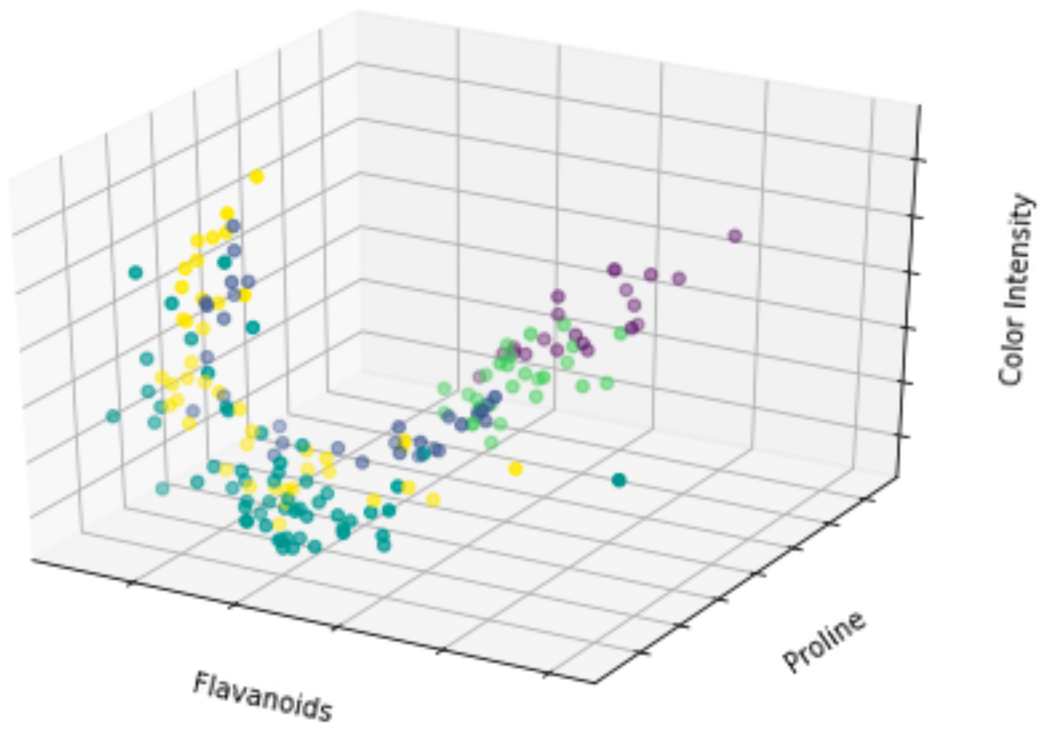


Running Agglomerative Clustering method against 5 clusters for the top 3 features as determined above. I start the timer before the fitting function and stop after the fitting function. Running the algorithm 5 times:

n clusters = 5

Run 1	0. 0008039359818212688
Run 2	0. 0008217980212066323
Run 3	0. 0008288479875773191
Run 4	0. 0008330359996762127
Run 5	0. 0011607060150709003

The average running time is about **0.8895 milliseconds** to fit 5 cluster. Comparing the running time against fitting 1 cluster above, the difference is very minimal. Intuitively, this makes sense because Agglomerative clustering works by having each point as its own cluster initially and find a pair of most similar clusters and merge basing on their distance (distance between two clusters). Furthermore, Agglomerative Clustering handles non-convex clustering better than K-Means.

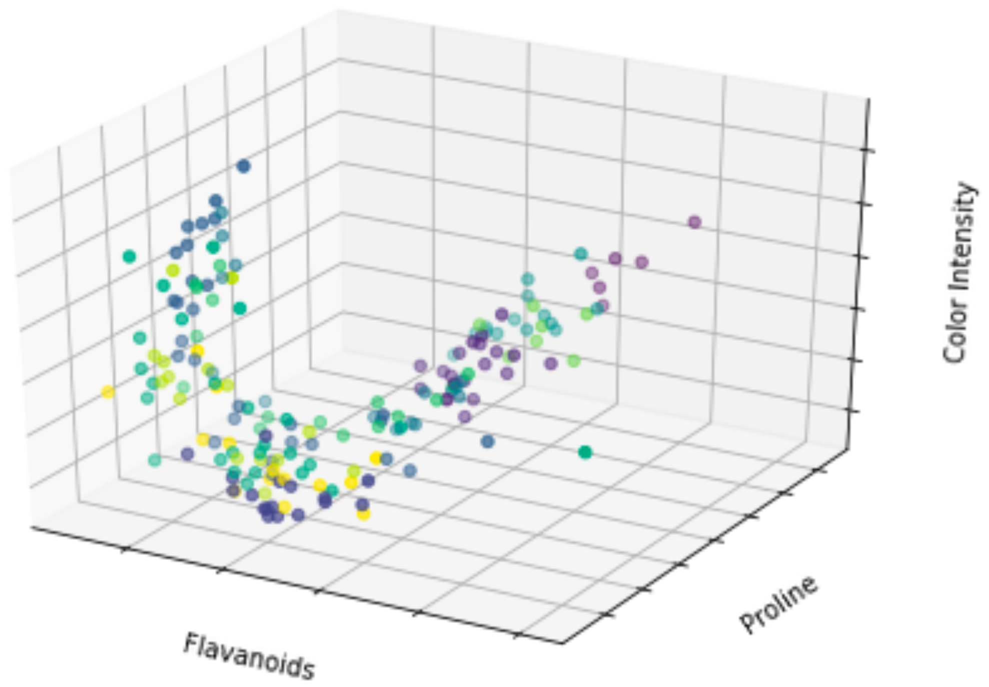


Running Agglomerative Clustering method against 11 clusters for the top 3 features as determined above. I start the timer before the fitting function and stop after the fitting function. Running the algorithm 5 times:

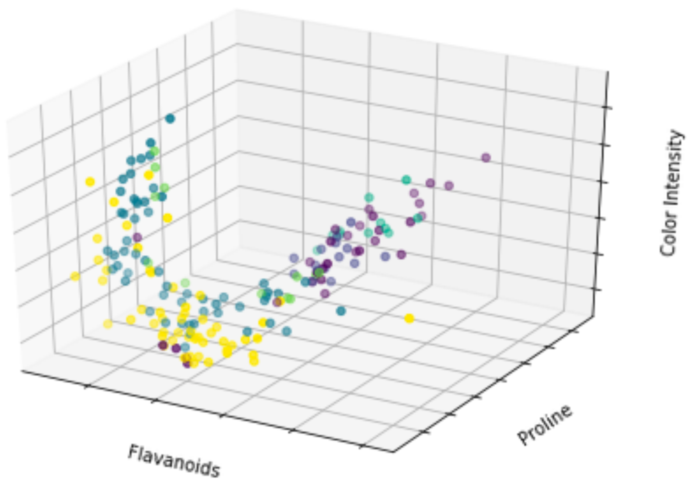
n clusters = 11

Run 1	0. 0008172669913619757
Run 2	0. 0008151039946824312
Run 3	0. 0008365200192201883
Run 4	0. 0008419170044362545
Run 5	0. 0008454299822915345

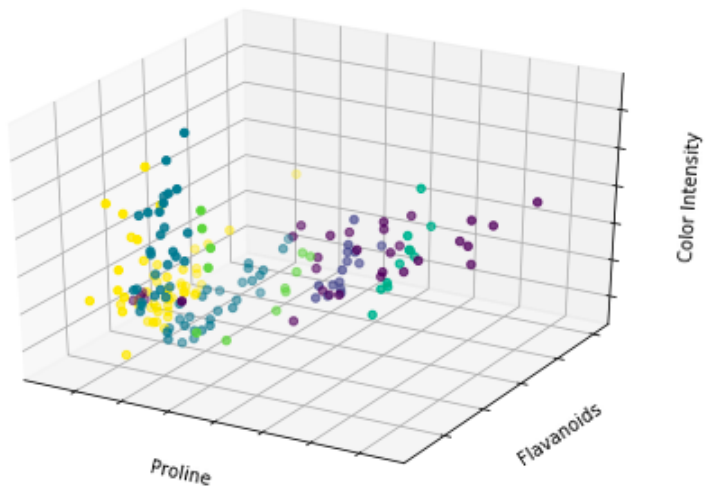
The average running time is about **0.83124 milliseconds** to fit 11 cluster. Again, the difference in average running time to fit 11 clusters is very minimal.



4) The nice thing about DBSCAN clustering method is that it does not require a defined number of clusters like in K-Means, Birch, and Agglomerative Clustering methods. However, it does require epsilon (radius from each point to neighbors) and the minimum neighbors in order to form clusters. However, choosing the correct epsilon and minimum sample can be difficult and important since it determines the number of clusters generated in the model. DBSCAN can also discover any arbitrary shapes.



The rule of thumb of choosing the minimum points can be derived from the number of dimensions D in the dataset, minPts >= D + 1. Since we have 3 dimensions in this dataset, any minimum sample above 4 would be ideal. Using Euclidean metric in this dataset, I have concluded that an epsilon value of 20, and minimum sample of 5 do generate the same 5 clusters 3D graph as above. Comparing the 3-D graphs from DBSCAN, Birch, and Agglomerative Clustering methods, I believe that having 5 clusters in this dataset is the most optimized number of clusters.



5) In conclusion, I used random forest method to determine the top three important features of Wine dataset which are Flavanoids, Proline, and Color Intensity. Using the top three features, I ran against popular clustering methods such as K-Means, Birch, Agglomerative Clustering, and DBSCAN.

   K-Means clustering method is the first method that is used to analyze for clusters because of its simplicity and easy to use. However, K-means do have some shortfalls such that it does not detect outliers, and it does not work well with non-convex shapes. In this dataset, which contains a nonconvex shape as seen in the graphs above, K-means method was unable to cluster the shape correctly. Furthermore, K-means does not scale with large datasets. Looking at the running time comparison above, having more clusters would take longer running time since K-Means needs compare the distance again for every point against the mean center of clusters until all points are satisfied with their mean cluster centers.

   BIRCH clustering method is more efficient than K-means in terms of running time. Since Birch only requires one data scan to build a CF tree and it does not require data all at once, this can reduce the running time greatly. However, there might be some running cost if the CF Tree requires splitting but this is minimal because the splitting would be done in memory. If new point is inserted, the algorithm can use depth search traverse to find the most optimal cluster to fit the point in thus avoiding having unnecessary rescans of data like in K-Means. It can also scale well with larger dataset. However, BIRCH do have some downfalls. The size of each node in CF tree can hold only a limited number of entries due to the size, and the method doesn’t perform well if the shape is not spherical because it uses radius or diameter to control the boundary of a cluster.

   Agglomerative Clustering method seems to be the fastest among K-Means and Birch, however it does not scale very well with large dataset due to its rather slow time complexity O(n^2) Agglomerative Clustering can produce an ordering of data points, which maybe informative for data display, and smaller clusters are generated which may be helpful for discovery. However, data points may be incorrectly grouped at early stage and there is no way to backtrack. Therefore, the result should be examined with closely consideration to ensure that the clustering makes sense. Furthermore, Agglomerative Clustering uses the distance between clusters to merge and the use of different distance metrics for measuring may generate different results. Therefore, Agglomerative may require multiple experiments with different metrics.

   DBSCAN does scale well with large datasets and able to discover clusters with any arbitrary shape. It does not require a specific number of clusters, and is robust to outliers. DBSCAN requires two parameters, epsilon (the radius from the core data point), and minimum number of neighbors. However, choosing a meaningful distance threshold epsilon can be difficult. DBSCAN cannot cluster datasets with large differences in densities since the minimum of neighbors and epsilon combination cannot be chosen appropriately for all clusters. A better method similar to DBSCAN is OPTICS which epsilon only serves the upper limit of the radius. The radius in OPTICS would be based on how dense the number of neighbors defined by the minimum neighbors given. The OPTICS method allows extraction of clusters with any arbitrary epsilon by calculating the core distance and reachability distance.