Gather data, determine the method of data collection and provenance of the data

In the earliest phase, select a data source and problem. Feel free to share and discuss your idea on the class discussion board.

For this project, I will be using data from Kaggle, which can be found at the link below

> https://www.kaggle.com/datasets/uciml/red-wine-quality-cortez-et-al-2009

Data provenance:

The data from Kaggle comes from the University of California, Irvine Machine Learning Repository, who got the data from the paper "Modeling wine preferences by data mining from physicochemical properties", which can be found at the link below

> https://repositorium.sdum.uminho.pt/bitstream/1822/10029/1/wine5.pdf

Data:

To summarize, the data describes a multitude of red wine sample's chemical properties, all of the variety of portuguese "Vinho Verde", which comes from the northwest region of Portugal, a wine producing are since the 1600's. In addition to the chemical composition, an additional data point is assigned to each wine sample, called quality. It is on a scale of 0-10, is stated in the research paper as being the median of multiple evaluates of that wine sampe. A full desciption of the data and it's source can be found in the research paper's Section 2.1.

Method of data collection:

As described in section 2.1 of the paper, the data was collected from a an offical wine certification entity called the Vinho Verde Viticulture Commision, or CVRVV. The CVRVV is an inter-professional body of farm and trade professionals and representitives from the vinho verde wine agricultural commodity chain, and has existed since its inception in 1926. The data was provided to the original researchers devoid of any unique identifiers that could link a sample to its unique year, vinyard, grape species, or other nationally protected clasifier. For more informnation on the region, see the link below.

> https://www.vinhoverde.pt/

Citation:

Cortez, P. et al. "Modeling wine preferences by data mining from physicochemical properties." Decis. Support Syst. 47 (2009): 547-553.

Identify an Unsupervised Learning Problem

Model building and training may depend on their data type(s) and task type(s). When using multiple models, at least one of them should be an unsupervised approach.

If you're going to use a Kaggle competition or similar, you must focus more on model building and/or analysis to be a valid project. Replicating what's in the Kaggle kernel or other notebooks available online is not a valid project. It is reasonable to add different approaches and compare them with the Kaggle kernel or other notebooks available online. It is also good to find a research paper, implement an algorithm, and run experiments comparing its performance to different algorithms. </p>

Using the wine data collected, I will investigate the following question:

- Is it possible, using unsupervised learning methods, to determine a wine sample's given quality value based solely on its chemical properties? i.e. is sample quality an indicator of a specific collection of chemical properties in the wine itself or is sample quality a more subjective value
- If so, to what degree of accuracy can a samples quality be predicted, and if not, what is/could be hindering the models ability to make accurate predictions

Exploratory Data Analysis (EDA) - Inspect, Visualize, and Clean the Data

Go through the initial data cleaning and EDA and judge whether you need to collect more or different data.

> Describe the factors or components that make up the dataset (The "factors" here are called "features" in the machine learning term. These factors are often columns in the tabulated data). For each factor, use a box-plot, scatter plot, histogram, etc., to describe the data distribution as appropriate.

In the dataset chosen, there are a total of 12 factors. they are:

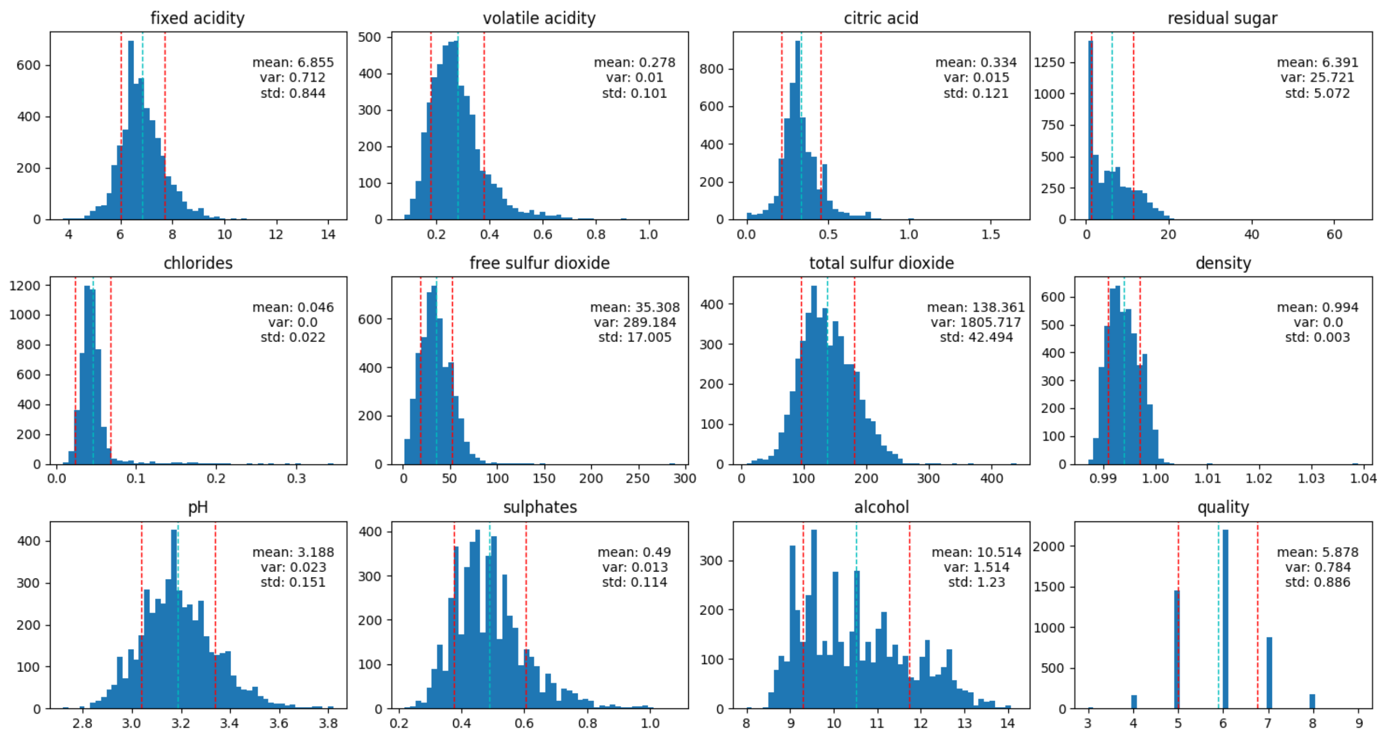
- fixed acidity
- volitile acidity
- citric acid
- residual sugar
- chlorides
- free sulfur dioxides
- total sulfur dioxide
- density
- pH
- sulphates
- alcohol
- quality

Below is a print out of the first 5 rows of the data and a assesment of data types and non-null counts for each feature

Out[34]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.0	0.27	0.36	20.7	0.045	45.0	170.0	1.0010	3.00	0.45	8.8	6
1	6.3	0.30	0.34	1.6	0.049	14.0	132.0	0.9940	3.30	0.49	9.5	6
2	8.1	0.28	0.40	6.9	0.050	30.0	97.0	0.9951	3.26	0.44	10.1	6
3	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9	6
4	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9	6

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4898 entries, 0 to 4897
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   fixed acidity         4898 non-null   float64
 1   volatile acidity      4898 non-null   float64
 2   citric acid           4898 non-null   float64
 3   residual sugar        4898 non-null   float64
 4   chlorides             4898 non-null   float64
 5   free sulfur dioxide   4898 non-null   float64
 6   total sulfur dioxide  4898 non-null   float64
 7   density               4898 non-null   float64
 8   pH                    4898 non-null   float64
 9   sulphates             4898 non-null   float64
 10  alcohol               4898 non-null   float64
 11  quality               4898 non-null   int64  
dtypes: float64(11), int64(1)
memory usage: 459.3 KB
```



note:

Cyan dashed lines: mean of distribution

Red dashed lines: mean +/- 1 std. dev.

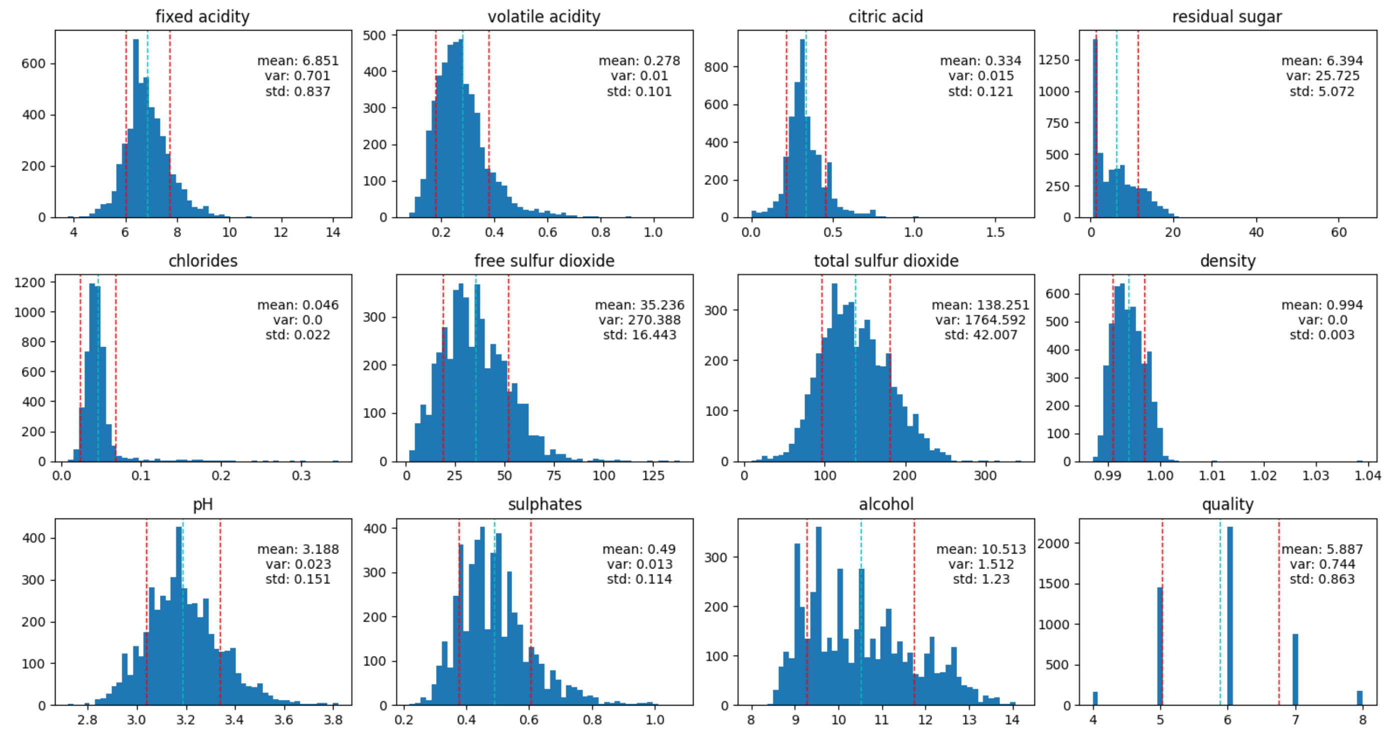
From these histograms, we can see that there are many instances of outliers for each feature.

Looking at the 'quality' feature in particular, we can se that there are many more counts of quality's of '5' and '6' than all the other counts combigned. We are going to use this fact to inform us on the numnber of bins/categories to use with each unsupervised learning models.

```
all_data:
row 0, quality == 3 count: 20
row 1, quality == 4 count: 163
row 2, quality == 5 count: 1457
row 3, quality == 6 count: 2198
row 4, quality == 7 count: 880
row 5, quality == 8 count: 175
row 6, quality == 9 count: 5
```

```
all_data:
row 0, quality == 3 count: 0
row 1, quality == 4 count: 163
row 2, quality == 5 count: 1457
row 3, quality == 6 count: 2198
row 4, quality == 7 count: 880
row 5, quality == 8 count: 175
row 6, quality == 9 count: 0
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 4873 entries, 0 to 4897
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   fixed acidity         4873 non-null   float64
 1   volatile acidity      4873 non-null   float64
 2   citric acid           4873 non-null   float64
 3   residual sugar        4873 non-null   float64
 4   chlorides             4873 non-null   float64
 5   free sulfur dioxide   4873 non-null   float64
 6   total sulfur dioxide  4873 non-null   float64
 7   density               4873 non-null   float64
 8   pH                    4873 non-null   float64
 9   sulphates             4873 non-null   float64
 10  alcohol               4873 non-null   float64
 11  quality               4873 non-null   int64  
dtypes: float64(11), int64(1)
memory usage: 494.9 KB
```



From the above displays, we can saee that there are each data point is either an integer or a decimal, there are no missing data points, and every feature has a consitant data type. We can say now that there is no need for data cleaning: everything is ready to go.

Given the nature of the data (floating point integers and straight integers), histograms will be effective at showing the distributions of each feature

---

Data cleaning and investigation of outliers in data

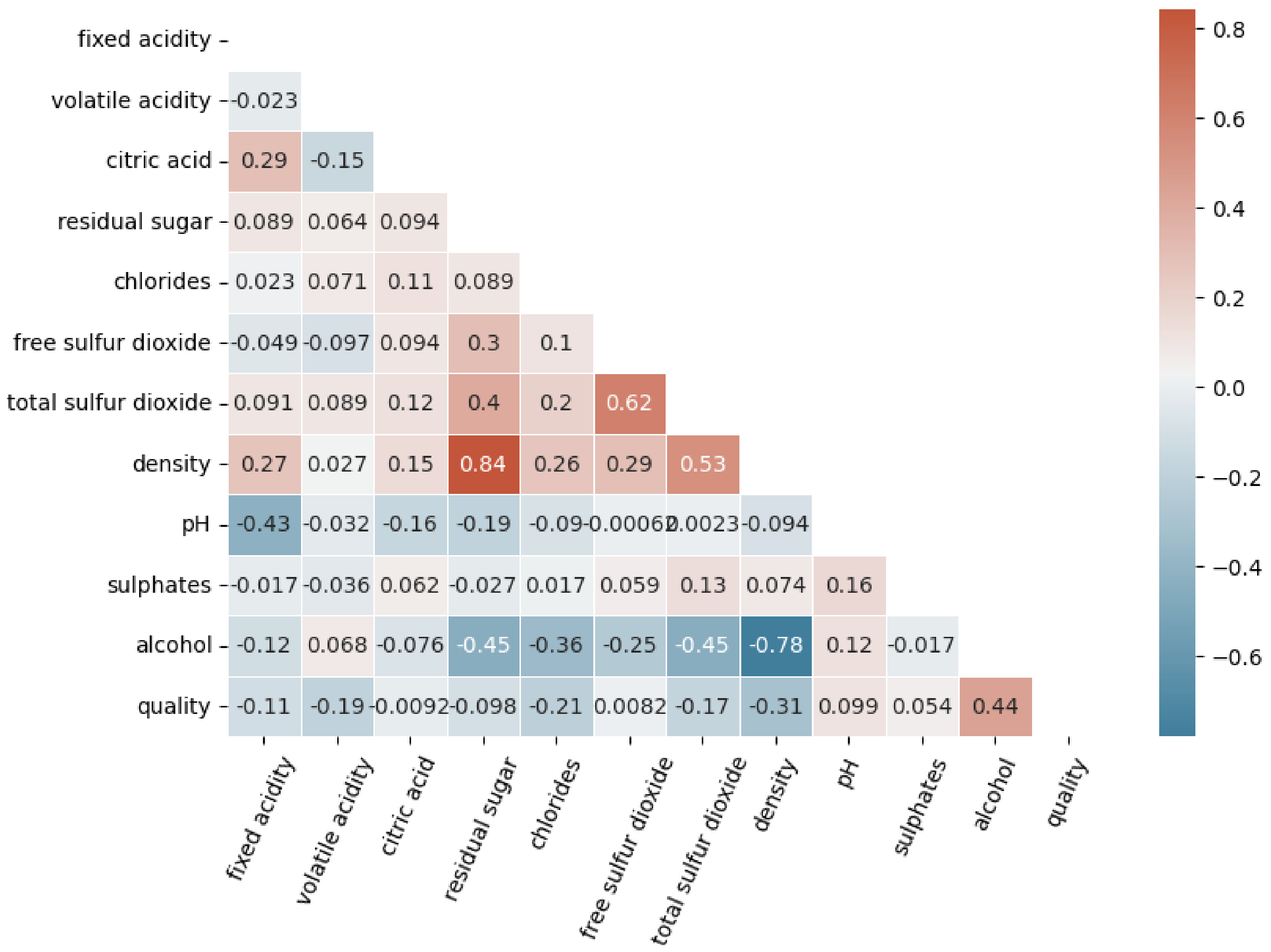
> Are there missing data values for specific factors? How will you handle the data cleaning? Will you discard, interpolate or otherwise substitute data values?

Right away, we can see that there is no need for data cleaning, as explained earlier.

To inspect the data for outliers, we can plot each feature as a histogram and look at the tails for each distribution.

> Describe correlations between different factors of the dataset and justify your assumption that they are correlated or not correlated. You may use numeric or qualitative/graphical analysis for this step.

To assess correlations between features in this data set, we're going to use a correlation matrix, generaetd using pandas' .corr() function and plotted using seaborn's heatmap()



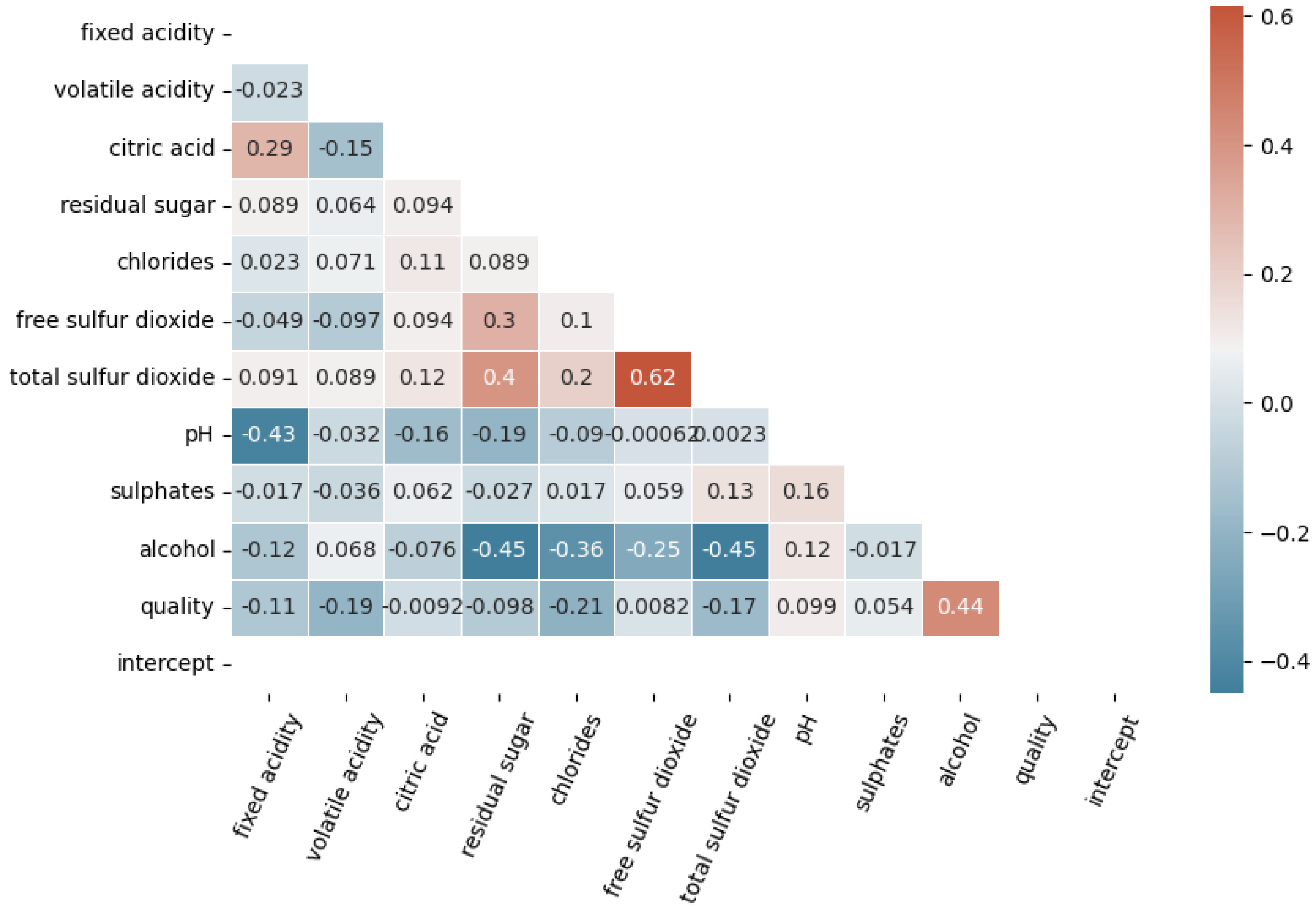
Out[409]:

	Variable	VIF
7	density	28.591234
3	residual sugar	12.947302
10	alcohol	7.807600
0	fixed acidity	2.696863
6	total sulfur dioxide	2.239495
8	pH	2.215431
5	free sulfur dioxide	1.795035
11	quality	1.392506
4	chlorides	1.236874
1	volatile acidity	1.203769
2	citric acid	1.165228
9	sulphates	1.147761

Out[9]:

	Variable	VIF
6	total sulfur dioxide	2.155565
9	alcohol	1.996363
5	free sulfur dioxide	1.756151
3	residual sugar	1.464873

10	quality	1.375037
0	fixed acidity	1.359289
7	pH	1.332002
4	chlorides	1.204405
1	volatile acidity	1.196604
2	citric acid	1.159905
8	sulphates	1.060653



In the above map, more transparent cells represent little correlation between features, solid cells represent strong correlation, and red and blue colors represent positive and negative correlations, respectively.

You can see that there are not many strong correlations one way or the other. Some strong correlations exist between:

- (fixed acidity, pH)
- (fixed acity, density)
- (free sulfur diozide, total sulfur dioxide)
- (citric acid, fixed acidity)
- (citric acid, volatile acidity)
- (density, alcohol)
- (quality, alcohol)

> Determine if any data needs to be transformed. For example, if you're planning on using an SVM method for prediction, you may need to normalize or scale the data if there is a considerable difference in the range of the data.

As stated previously, we are going to look at models and results from K-means, Non-zero Matrix Factorization,

Out[40]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates
count	4873.000	4873.000	4873.000	4873.000	4873.000	4873.000	4873.000	4873.000	4873.000	4873.000
mean	6.851	0.278	0.334	6.394	0.046	35.236	138.251	0.994	3.188	0.490
std	0.837	0.101	0.121	5.073	0.022	16.445	42.011	0.003	0.151	0.114
min	3.800	0.080	0.000	0.600	0.009	2.000	9.000	0.987	2.720	0.220
25%	6.300	0.210	0.270	1.700	0.036	23.000	108.000	0.992	3.090	0.410
50%	6.800	0.260	0.320	5.200	0.043	34.000	134.000	0.994	3.180	0.470
75%	7.300	0.320	0.390	9.900	0.050	46.000	167.000	0.996	3.280	0.550
max	14.200	1.100	1.660	65.800	0.346	138.500	344.000	1.039	3.820	1.080

> Using your hypothesis, indicate if it's likely that you should transform data, such as using a log transform or other transformation of the dataset.

> If you believe that specific factors will be more important than others in your analysis, you should mention which and why. You will use this to confirm your intuitions in your final write-up.

Perform Analysis Using Unsupervised Learning Models of your Choice, Present Discussion, and Conclusions

Model building and training may depend on their data type(s) and task type(s). Depending on your project, you may have one model or more. Generally, it is deemed a higher quality project if you compare multiple models and show your understanding of why specific models work better than the other or what limitations or cautions specific models may have. **When using multiple models, at least one of them should be an unsupervised approach**.

For machine learning models, another recommendation is to show enough effort on the hyperparameter optimization.

If your project involves making a web app (not required), you can include the demo.</p>

Building and training of unsupervised learning models

Generating, train, validation, and testing data

```
all:
row 0, quality == 3 count: 0
row 1, quality == 4 count: 163
row 2, quality == 5 count: 1457
row 3, quality == 6 count: 2198
row 4, quality == 7 count: 880
row 5, quality == 8 count: 175
row 6, quality == 9 count: 0

y_train:
row 0, quality == 3 count: 0
row 1, quality == 4 count: 75
row 2, quality == 5 count: 719
row 3, quality == 6 count: 1126
row 4, quality == 7 count: 441
row 5, quality == 8 count: 87
row 6, quality == 9 count: 0

y_validate:
row 0, quality == 3 count: 0
row 1, quality == 4 count: 48
row 2, quality == 5 count: 374
row 3, quality == 6 count: 511
row 4, quality == 7 count: 230
row 5, quality == 8 count: 43
row 6, quality == 9 count: 0

y_test:
row 0, quality == 3 count: 0
row 1, quality == 4 count: 40
row 2, quality == 5 count: 364
row 3, quality == 6 count: 561
row 4, quality == 7 count: 209
row 5, quality == 8 count: 45
row 6, quality == 9 count: 0


(2448, 11)
(1219, 11)
(1219, 11)
2438
0.9893355209187858
2.0082034454470876
0.4946677604593929
1.0041017227235438
540     4
4233    6
3774    5
2746    6
1970    6
Name: quality, dtype: int64
```

Model selection

To try and find an interesting set of models, only the models that meet the following critera will be considered 'optimal':

1. achieved the highest accuracy score
2. had correctly predicted at least one sample's 'quality' for each value of 'quality' (i.e. every category had at least one count)

This way, a models overall accuracy score would not exclusively determine what was a good model and the testing would not return models that predicted almost every label to fall into the largest possible category.

Model set 1: K-means

Here we train and rank K-means models, selecting only models that meet the critera above.

```
array([1, 1, 0, 0, 0])
[False False  True  True  True]
3
array([0, 0, 0])
array([0, 1, 1])
[ True False False]
0.3333333333333333
[0.1  0.1  0.1  0.1  0.09]
[True, True, True, True, False]
4
```

```
% done: 100.0, % no model: 25.0, frac: 0.253
```

```
% done: 100.0
```

```
% done: 100.0, % no model: 25.0, frac: 0.253
```

Model set 2: Aglomerative Clustering

Here we train and rank Agglomerative Clustering models, selecting only models that meet the critera above.

```
% done: 100.0, % no model: 43.75, frac: 0.4375
```

```
% done: 100.0, % no model: 6.25, frac: 0.0625
```

```
% done: 100.0, % no model: 6.25, frac: 0.0625
```

Model set 3: Non-zero Matrix Factorization (NMF)

Here we train and rank NMF models, selecting only models that meet the critera defined earlier.

	0	1	2	3	4	5	6

count	2448.000000	2448.000000	2448.000000	2448.000000	2448.000000	2448.000000	2448.000000	2448.00000
mean	8.231728	2.835525	2.676520	1.272004	2.125261	2.140425	3.305588	339.90799
std	1.000199	1.000204	1.000169	1.000171	1.000197	1.000198	1.000217	1.00021
min	4.695000	0.822000	0.000000	0.119000	0.418000	0.180000	0.239000	337.54100
25%	7.584000	2.159000	2.148000	0.339000	1.672000	1.439000	2.607000	339.12025
50%	8.185000	2.673000	2.545000	1.016000	1.997000	2.039000	3.204000	339.82900
75%	8.787000	3.290000	3.102000	2.003500	2.322000	2.819000	3.994000	340.62550
max	17.093000	10.331000	13.204000	6.292000	16.071000	7.676000	8.226000	345.47100

```
% done: 100.077

% done: 100.0, % no model: 96.5608, frac: 0.965656

% done: 100.0, % no model: 94.3122, frac: 0.943139

% done: 100.0, % no model: 95.3704, frac: 0.953736

% done: 100.077

% done: 100.0, % no model: 98.6772, frac: 0.986887

% done: 100.0, % no model: 95.5026, frac: 0.955559

% done: 100.0, % no model: 97.2222, frac: 0.972222

% done: 100.077

% done: 100.0, % no model: 99.0741, frac: 0.990707

% done: 100.0, % no model: 95.3704, frac: 0.953736

% done: 100.0, % no model: 99.3386, frac: 0.993434
```

> what goes in:
> - the predicted categories for each sample (pred_cat_in) and the true categories for each sample (true_cat_in)
>
> What happens in the program:
>
> 1. The predicted category sample data and the true category data are converted into dataframes
>   - each dataframe has an additional column 'index', where the samples original index from the training data is stored. This is used to pair the two dataframes in the next step
> 2. The dataframes are fed into a nested for loop
>   A. In the top loop, the "true_cat_in" dataframe is reduced such that only 'quality_true'==i, for each category 'quality_true'={0,1,2,3}, is present.
>   B. In the inner for loop, the "pred_cat_in" dataframe is reduced such that only 'quality_pred'==j, for each category 'quality_pred'={0,1,2,3}, is present.
>   C. The two reduced dataframes are then merged on the 'index' column, to see for each ('quality_true' == i, 'quality_pred' == j), which samples that have the true category 'quality_true' == i are present in 'quality_pred' == j

> what comes out:
> - a matrix of dimension m, n = (n_components, n_components), where element i, j is the proportion of samples with true category i present in predicted category j

---

Discussion of model results

Note: Many graphs of confusion matrices were generated from the models trained above. The text at the top of this section contains the analysis and commentary. References to the charts are made to illustrate points.

Analysis:

Essentailly every model had difficulty finding an optimum catigorization that matched what the dataset defined as 'good' or 'bad' wines. When assessing what the predicted results were using accuracy scores and confusion matricies, you find a wide range of models that have very high accuracy scores but predict almost every label to be of one value (almost all predicted vallues fall in one of _n bins), or models with lower accuracy scores but have label counts that are closer to the counts of true labels.

Based on accuracy scores, the best performing unsupervised learning models, for all training, validating cases, were the Non-zero Matrix Factorization models. Technically, the highest accuracy agglomerative clustering model for n=2 outperformed the NMF model for n=2, but the model put every label but one into the most populus category, while the NMF model put all but two into the most populus category. The models that did the best maximizing the number of correct predictions for each bin, however, were the K-means models. To say another way, if 10 samples belonged to quality '5' and the available bins were '4', '5', and '6', the K-means algoithms would place 4 samples in '5', 3 samples in '3' and 3 samples in '6'. In no case was a k-means algorithm able to place more than half of the samples of a given quality into the correct label.

Each of the best performing models relyed on a different set of initialization parameters. Unfortunately, none of the best performing models were able to predict every category very well, they were only able to predict the most numerous categories well, which biased the accuracy results in their favor.

As the number of cluster/categories decreased, the scores for the best models increased significantly, which is not suprising. With fewer categories to predict, models can be found that place the most samples into one bin and find the relationship that connects them.

Discussion:

As stated earlier, the goal of this project was to investigate the question:

> Is it possible to group samples into categories that match quality values

Even though they did not score the absoulte highest accuracy scores, the only unsupervised models that could categorize samples into groups that matched their given quality values were the k-means algorithms. This hints that there could be a connection between quality and wine chemical properties beyond subjective preference and wine drinkers saying they like a wine because their friends or idols like the wine. In other words, it's not entirely random. Since these groupings are found without associating the known quality values with their samples, using supervised learning methods would likely produce accurate models that could better predict a samples quality value.

Why was k-means able to find similar groupings to the input quality data while the Agglomerative clustering and NMF methods did not?

As stated on sklearns website regarding clustering methods, Agglomerative has what they call a “rich get richer” behavior, meaning that the more heavily weighted a cluster center is, the more credence it is given over other clsuter centers. That leads to data points getting clustered with centers that have higher cluster densities, and can obscure any other possible linkages to other cluster centers. Some preprocessing could improve the models, such as normalization of each sample and standardizing each feature to have a mean of zero and standard deviation of 1.

With NMF, the shape and scale of the input data can have a large effect on the resulting predictions. If the data is not scaled properly, then the matrix factorization applied to the input data weight one feature over another in an artificial way, leading to obscuring the relationships between sample features and more inaccurate predictions. To account for this, the input data was standardized to have a standard deviation of 1 according to features, but additional steps could be taken. As stated about agglomerative clustering, the input data could benifit from additional preprocessing, such as normalization of each sample.

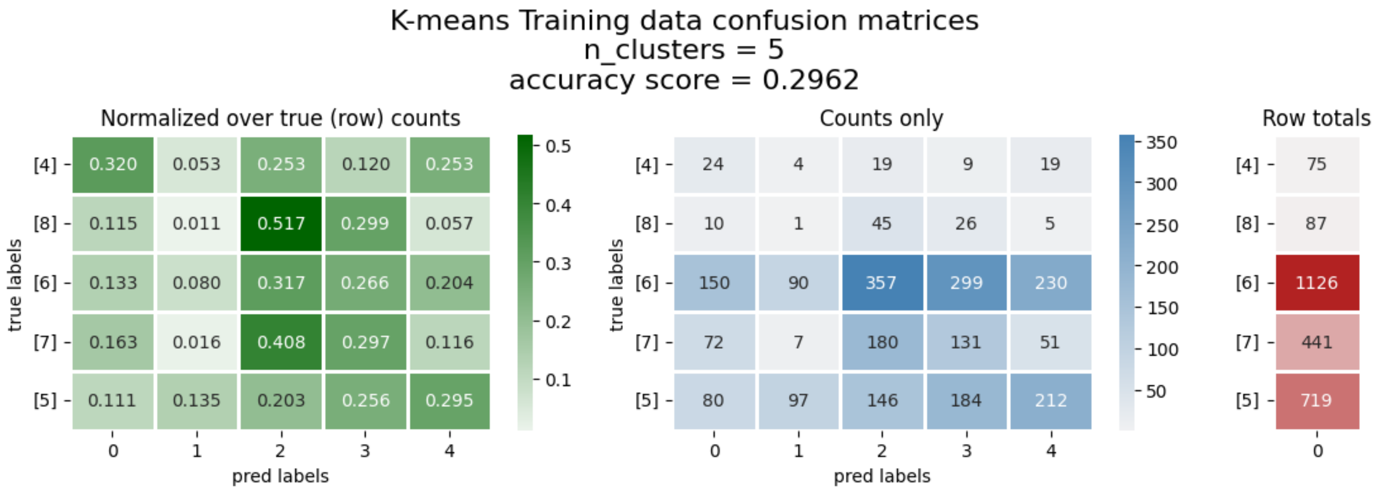
Charts:

K-means clustering

The confusion matricies and accuracy scores for best performing K-means models are displayed below. Below is a print out of the confusion matrix and and score for n_clusters=7 models, broken down by training, validating, and testing data sets

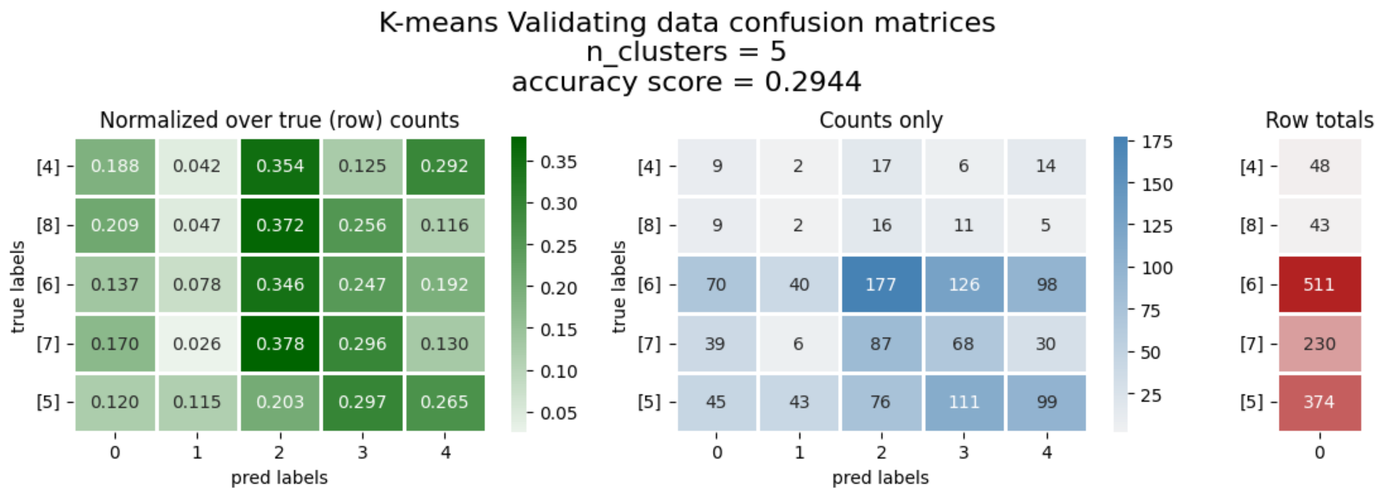
n_clusters = 5

```
K_means_Training_data_confusion_matrices_n_clusters___5_accuracy_score___0_2962
```

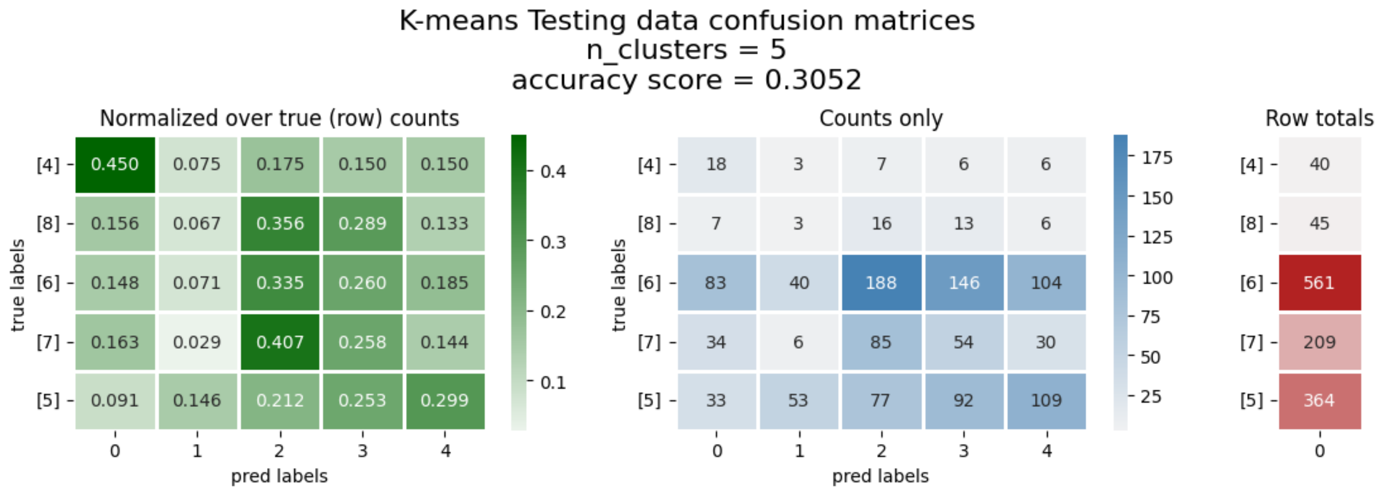


```
Algorithm: lloyd
Init: k-means++
model: KMeans(n_clusters=5, random_state=0)
score: 0.29616013071895425
map true label to pred label: {4: 0, 5: 4, 6: 2, 7: 3, 8: 1}
```

```
K_means_Validating_data_confusion_matrices_n_clusters___5_accuracy_score___0_2944
```



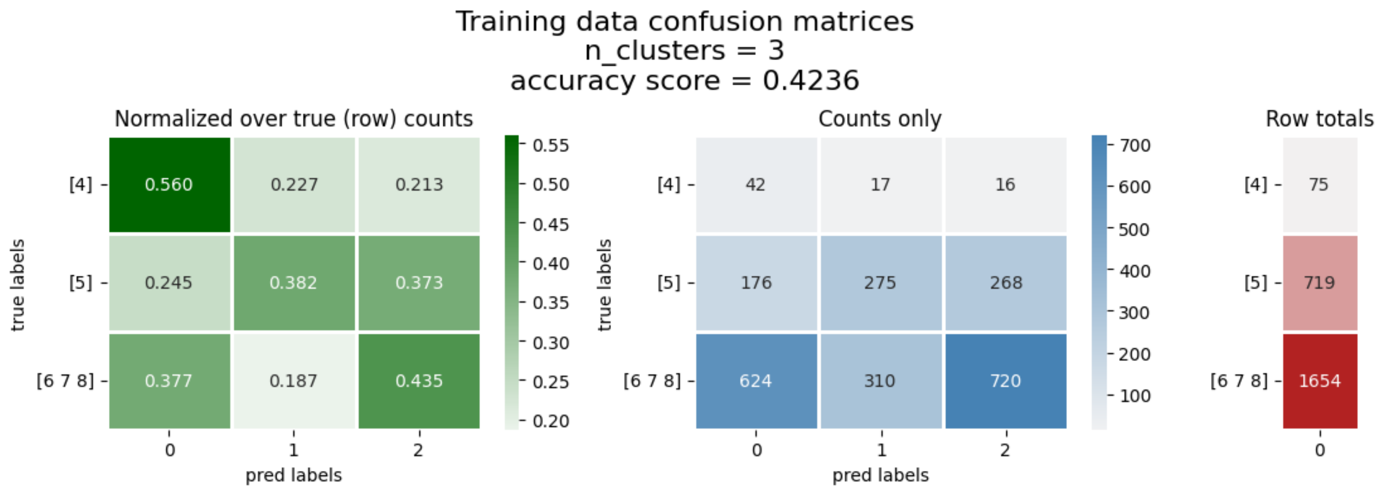
```
model: KMeans(n_clusters=5, random_state=0)
score: 0.2943615257048093

K_means_Testing_data_confusion_matrices_n_clusters___5_accuracy_score___0_3052
```



```
model: KMeans(n_clusters=5, random_state=0)
score: 0.3051681706316653
```

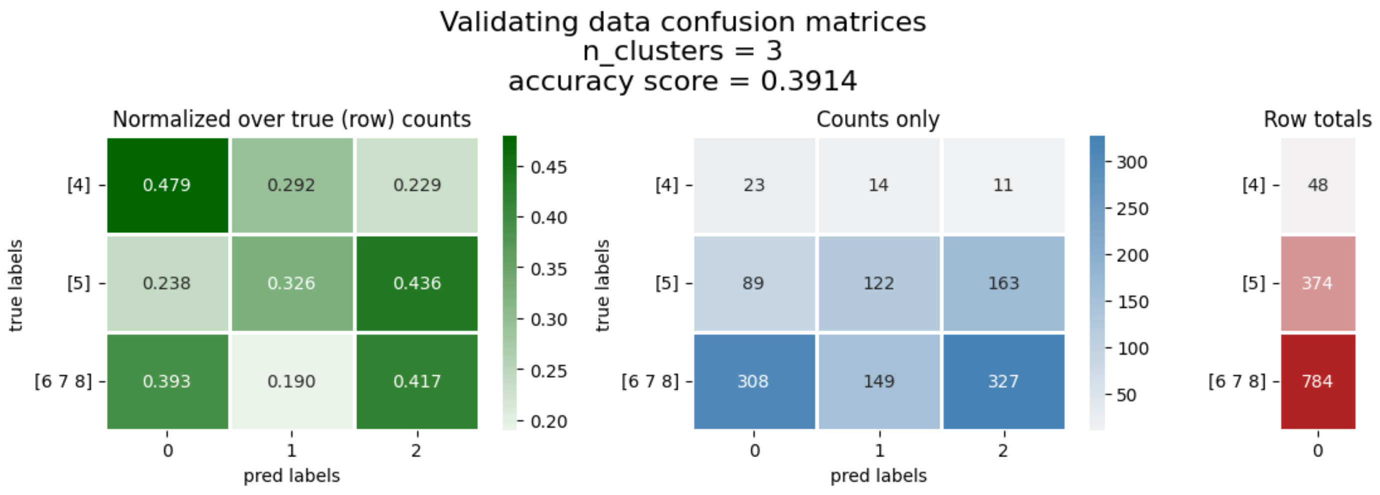
K-means clustering

n_clusters = 3

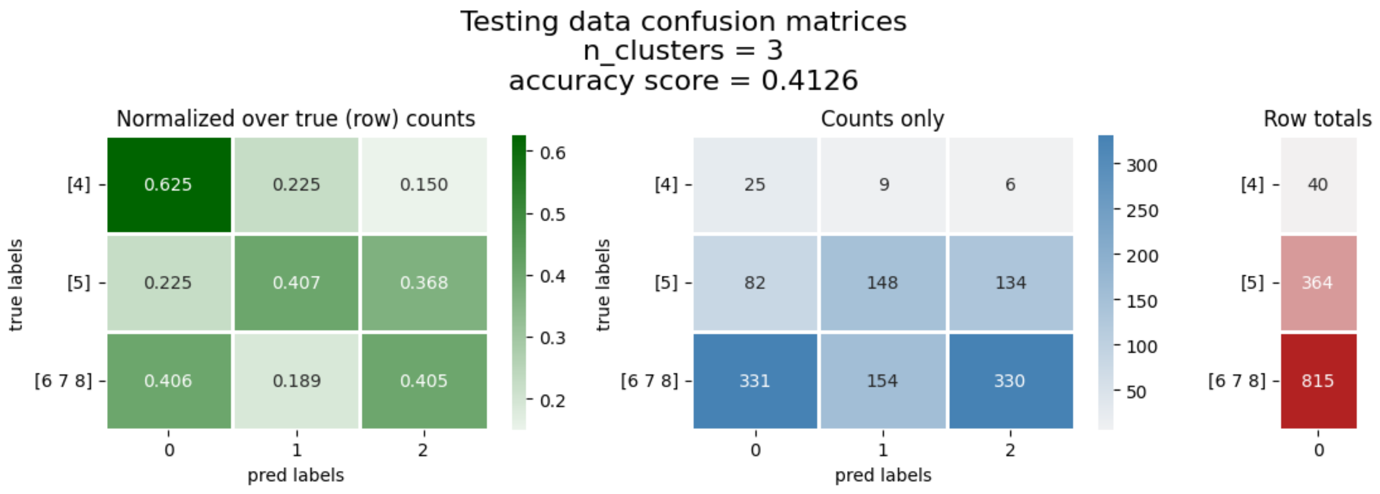


```
Algorithm: lloyd
Init: k-means++
model: KMeans(n_clusters=3, random_state=0)
score: 0.4236111111111111
```

```
map true label to pred label: {4: 0, 5: 1, 6: 2, 7: 2, 8: 2}
```



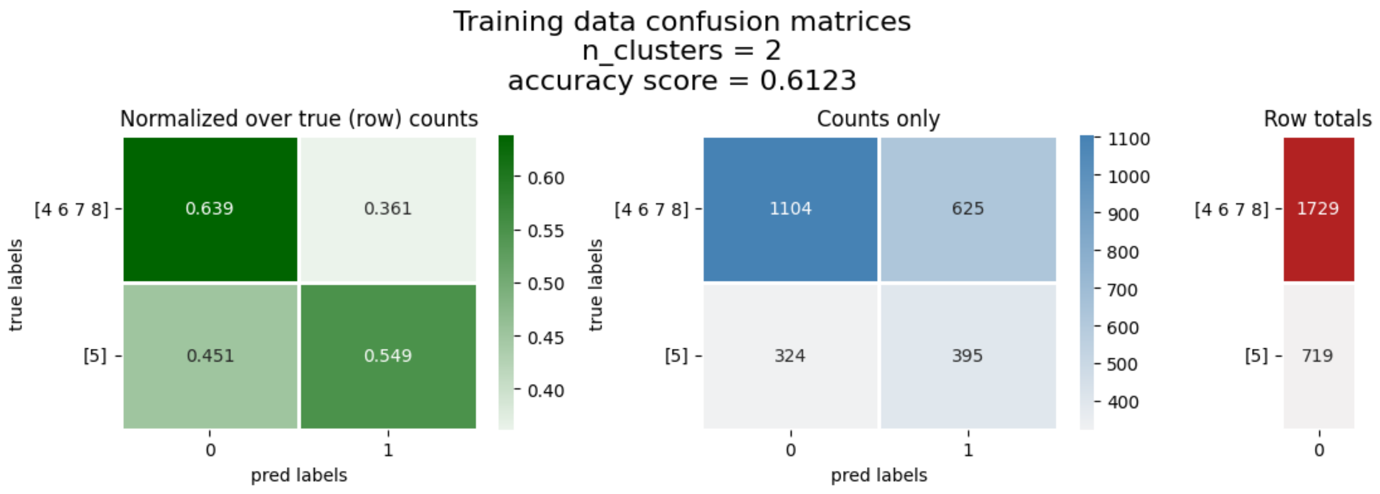
```
model: KMeans(n_clusters=3, random_state=0)
score: 0.3913764510779436
```



```
model: KMeans(n_clusters=3, random_state=0)
score: 0.41263330598851516
```

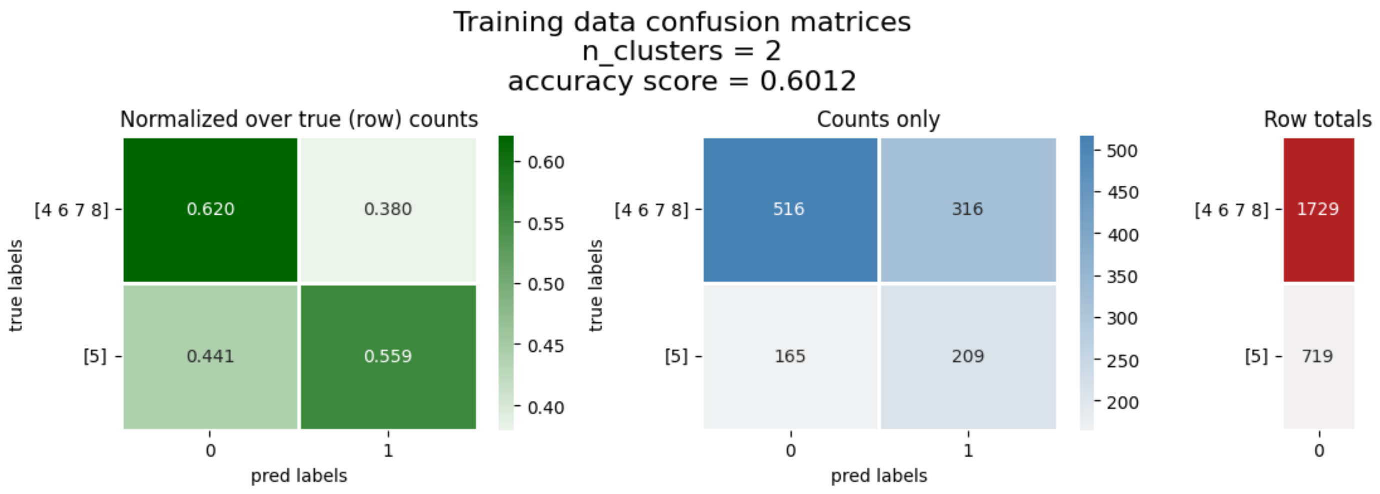
K-means clustering

n_clusters = 2

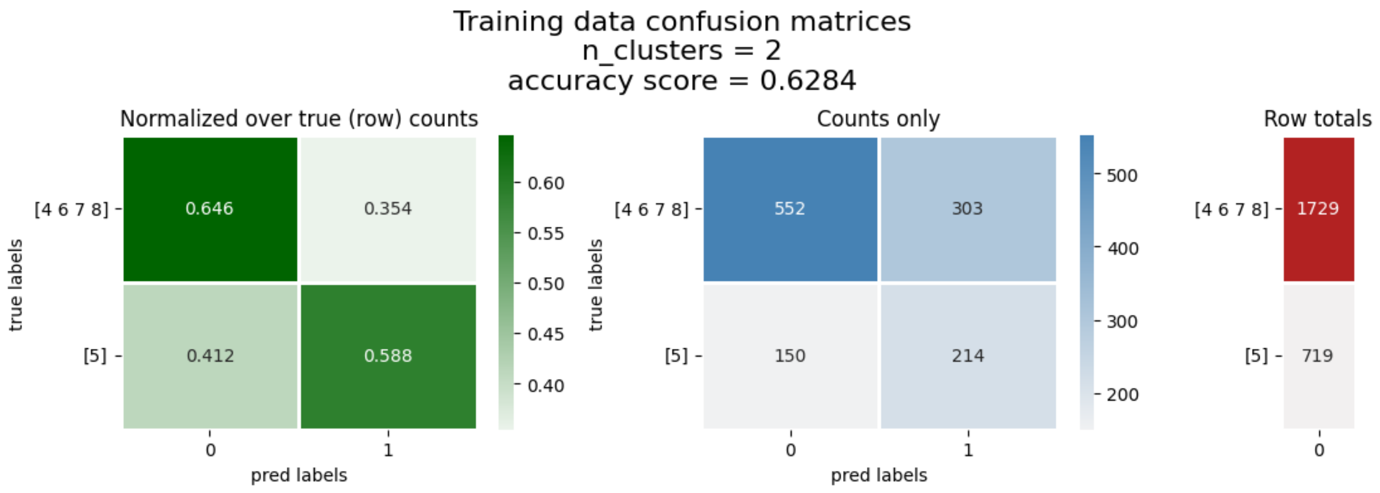


```
Algorithm: lloyd
Init: k-means++
model: KMeans(n_clusters=2, random_state=0)
score: 0.6123366013071896
```

```
map true label to pred label: {4: 0, 5: 1, 6: 0, 7: 0, 8: 0}
```



```
model: KMeans(n_clusters=2, random_state=0)
score: 0.6011608623548922
```



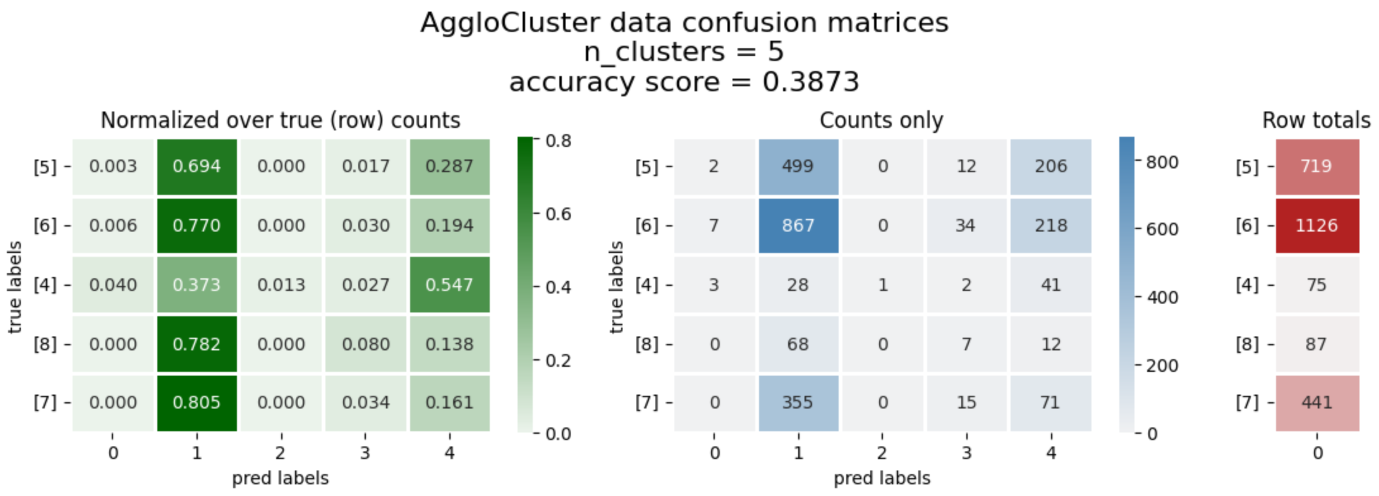
```
model: KMeans(n_clusters=2, random_state=0)
score: 0.6283839212469237
```

Agglomerative Clustering

The confusion matricies and accuracy scores for best performing Agglomerative Clustering models are displayed below. Below is a print out of the confusion matrix and and score for n_clusters=7 models

n_clusters = 5, 3, 2

```
AggloCluster_data_confusion_matrices_n_clusters___5_accuracy_score___0_3873
```



```
Algorithm: complete
Init: cosine
model: AgglomerativeClustering(linkage='complete', metric='cosine', n_clusters=5)
score: 0.3872549019607843
map true label to pred label: {4: 2, 5: 0, 6: 1, 7: 4, 8: 3}
```

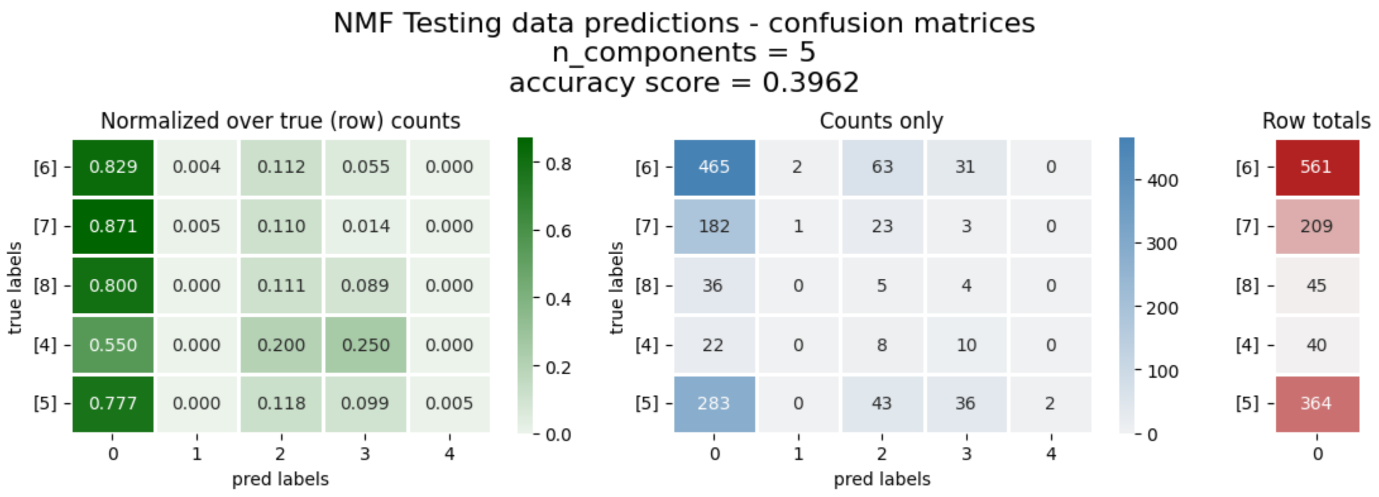
Non-zero Matrix factorization (NMF)

The confusion matricies and accuracy scores for best performing NMF models are displayed below. Below is a print out of the confusion matrix and and score for n_clusters=7 models, broken down by training, validating, and testing data sets

NMF

n_clusters = 5

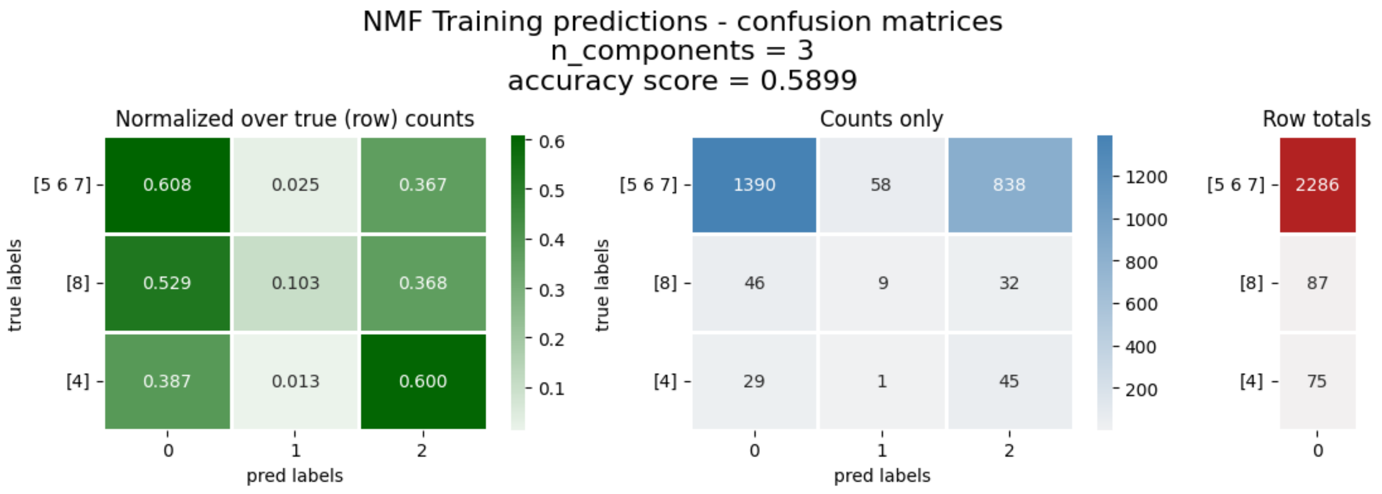
```
NMF_Testing_data_predictions___confusion_matrices_n_components___5_accuracy_score___0_39
62
```



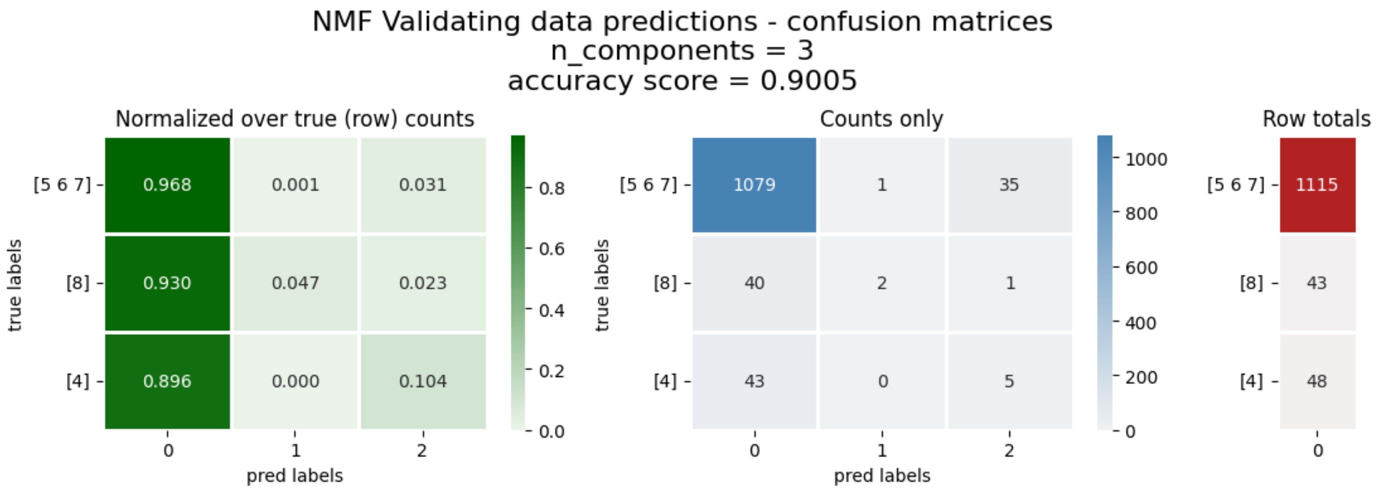
```
model params: {'alpha_H': 0.1, 'alpha_W': 0.1, 'beta_loss': 'frobenius', 'l1_ratio': 0.2
5, 'solver': 'mu'}
model: NMF(alpha_H=0.1, alpha_W=0.1, init='nndsvda', l1_ratio=0.25, n_components=5,
    random_state=1, solver='mu')
score: 0.39622641509433965
map true label to pred label: {4: 3, 5: 4, 6: 0, 7: 1, 8: 2}
```

NMF

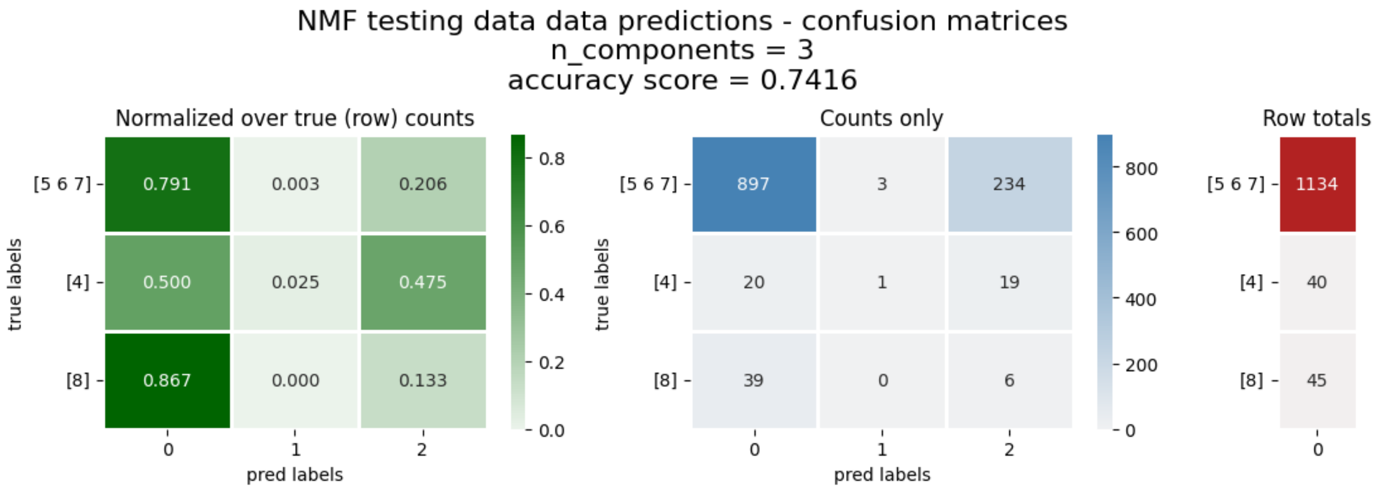
n_clusters = 3



```
model params: {'alpha_H': 0.1, 'alpha_W': 0.0001, 'beta_loss': 'kullback-leibler', 'l1_r
atio': 1, 'solver': 'mu'}
model: NMF(alpha_H=0.1, alpha_W=0.0001, beta_loss='kullback-leibler', init='nndsvda',
    l1_ratio=1, n_components=3, random_state=1, solver='mu')
score: 0.5898692810457516
map true label to pred label: {4: 2, 5: 0, 6: 0, 7: 0, 8: 1}
```



```
model params: {'alpha_H': 0.1, 'alpha_W': 0.1, 'beta_loss': 'frobenius', 'l1_ratio': 0.1
25, 'solver': 'mu'}
model: NMF(alpha_H=0.1, alpha_W=0.1, init='nndsvda', l1_ratio=0.125, n_components=3,
    random_state=1, solver='mu')
score: 0.900497512437811
map true label to pred label: {4: 2, 5: 0, 6: 0, 7: 0, 8: 1}
```

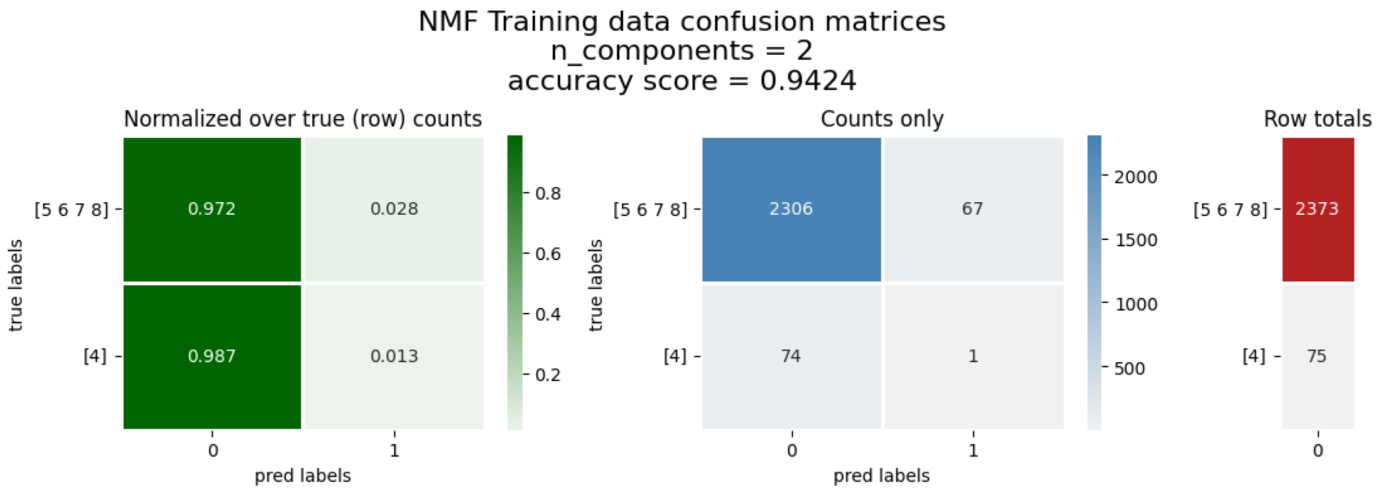


```
model params: {'alpha_H': 0.1, 'alpha_W': 0.0001, 'beta_loss': 'kullback-leibler', 'l1_r
atio': 0.5, 'solver': 'mu'}
```

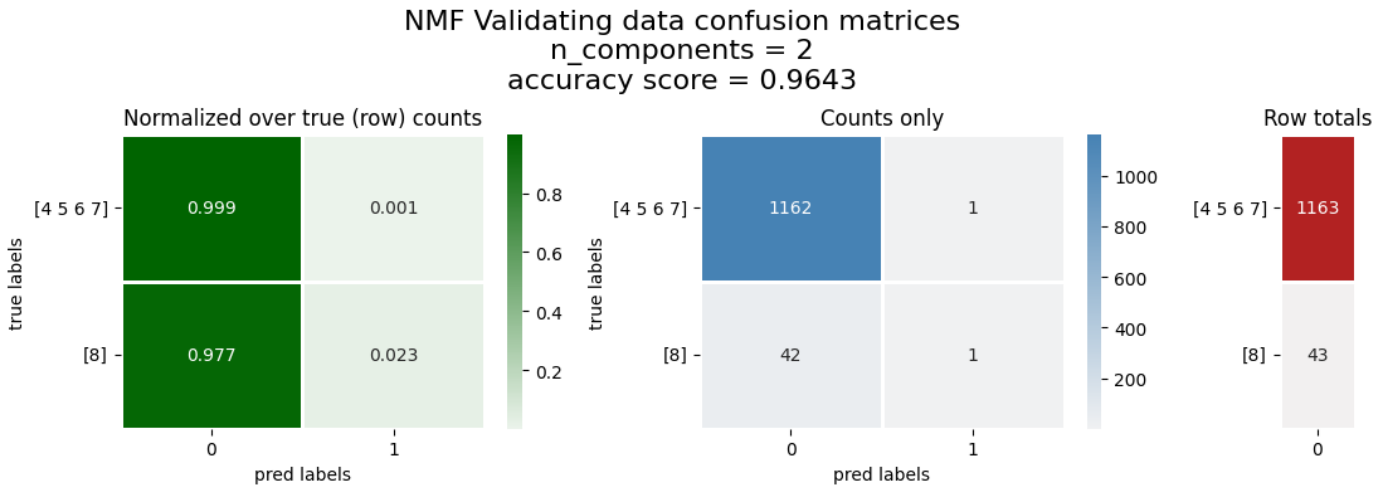
```
model: NMF(alpha_H=0.1, alpha_W=0.0001, beta_loss='kullback-leibler', init='nndsvda',
    l1_ratio=0.5, n_components=3, random_state=1, solver='mu')
score: 0.7415914684167351
map true label to pred label: {4: 1, 5: 0, 6: 0, 7: 0, 8: 2}
```

NMF

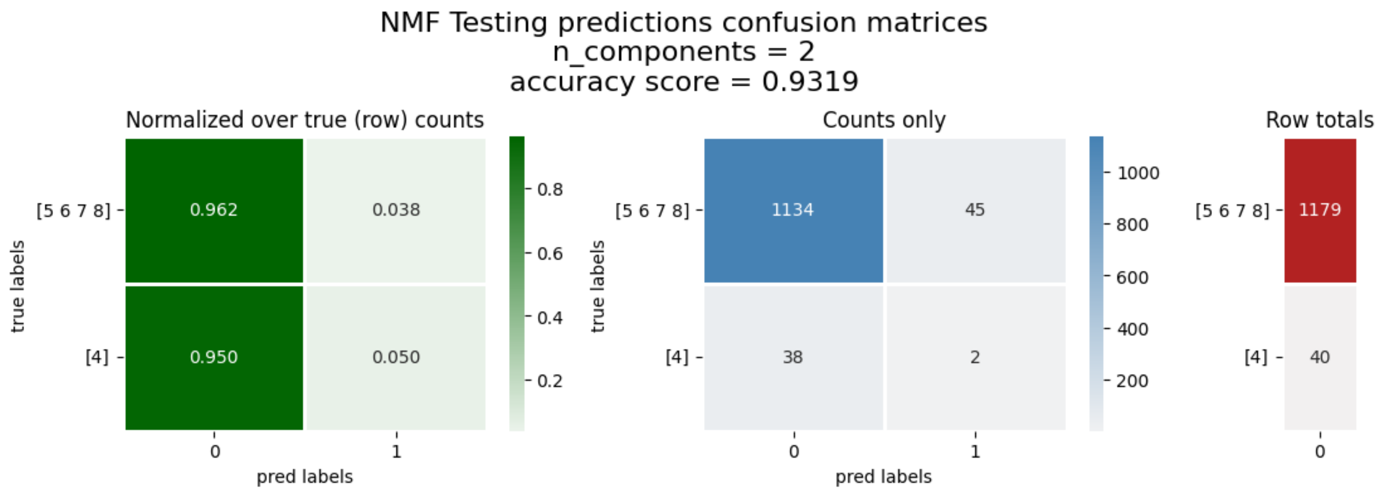
n_clusters = 2



```
model params: {'alpha_H': 0.1, 'alpha_W': 1e-05, 'beta_loss': 'kullback-leibler', 'l1_ra
tio': 0.875, 'solver': 'mu'}
model: NMF(alpha_H=0.1, alpha_W=1e-05, beta_loss='kullback-leibler', init='nndsvda',
    l1_ratio=0.875, n_components=2, random_state=1, solver='mu')
score: 0.9424019607843137
map true label to pred label: {4: 1, 5: 0, 6: 0, 7: 0, 8: 0}
```



```
model params: {'alpha_H': 0.1, 'alpha_W': 0, 'beta_loss': 'frobenius', 'l1_ratio': 0.25,
'solver': 'cd'}
model: NMF(alpha_H=0.1, alpha_W=0, init='nndsvda', l1_ratio=0.25, n_components=2,
    random_state=1)
score: 0.9643449419568823
map true label to pred label: {4: 0, 5: 0, 6: 0, 7: 0, 8: 1}
```



```
model params: {'alpha_H': 0.1, 'alpha_W': 1e-05, 'beta_loss': 'kullback-leibler', 'l1_ra
tio': 0.875, 'solver': 'mu'}
model: NMF(alpha_H=0.1, alpha_W=1e-05, beta_loss='kullback-leibler', init='nndsvda',
    l1_ratio=0.875, n_components=2, random_state=1, solver='mu')
score: 0.9319114027891715
map true label to pred label: {4: 1, 5: 0, 6: 0, 7: 0, 8: 0}
```

Conclusions

Based on this project, when using the K-means algorithm, it is possible to generate unsupervised learning models that *do* find sample clusters that group samples by their 'quality' values. These groupings are significant because the models generated here are not 'trained' to associate a given samples collective features to a specific value, they are only finding groupings based on each samples internal characteristics. This hints that there could be non-random statistical correlations between a wine's chemical properties and a wine sample's subjective quality. However, the fact that only one method of unsupervised learning, based on the k-means clustering methods, indicates that