Wine Review and recommendation

Krishna D.B. Anapindi, Individual Project, STAT 542

Introduction

In this project, I will be performing an in-depth analysis of wine-reviews and build a model to predict the review points given to each wine given certain attributes of the wine such as the description of the wine, the price per bottle, region/ country from which the wine has been produced, the winery from which the wine is produced and the taster who reviewed the wine. Since the wine ratings are a continuous distribution, (though not strictly continuous as the ratings only take integer values, for the purpose of this study, we assume them to be continuous) I will build regression models to see whether I can predict the ratings. To begin with, I start with a baseline model i.e. a linear regression to see how well the chosen covariates capture the ratings. further I plan to test two model:

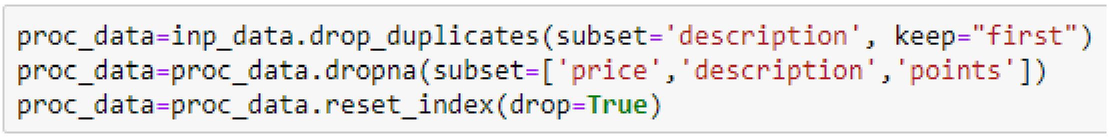
- A **Ridge regression**-which is a simple yet powerful model that performs L2 regularization on the β values.
- A more powerful gradient-boosting algorithm based on decision trees; the **CatBoost** which is designed to handle categorical data. Since the current dataset has a mix of both continuous and categorical data, I have specifically chosen this approach.

Finally, using these approaches, I was able to successfully model the reviews of a wide range of wines.

The second part of the project deals with coming up with recommendation of wineries to a customer based on his/her specific interests in Pinot Noir wines with a fruity flavor and under $20. For this, I make use of a dual-ranking approach. First, I rank all the wineries based on their similarity to the customer's description of a fruity wine. Secondly, I rank all the wineries based on their ratings (considering only pinot noir). Next, I sum up both the ranks and then choose the top 5 wineries that have the best ranking.

Part 1- Data Processing

For data processing, initially I remove the duplicate values (based on the review 'description'. Once this is performed, I then remove any null values present. For null values, I only consider the **price**, **description** and **points**.



```
proc_data=inp_data.drop_duplicates(subset='description', keep="first")
proc_data=proc_data.dropna(subset=['price','description','points'])
proc_data=proc_data.reset_index(drop=True)
```

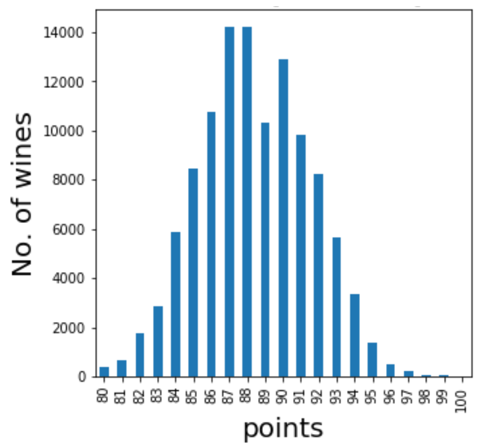
Since the description plays a major role in deciding the rating of a wine, they have to be incorporated into our model. There are several different ways available to convert textual data into numeric vectors. Here, I will be using the **Global Vectors for Word Representation (GloVe)** algorithm. GloVe is an unsupervised learning algorithm that is used to obtain vector representation of words. The training is performed on an aggregated global word-word co-occurrence statistic from the entire Wikipedia 2014 database. I used a constant vector size (100) to represent each word. The vectors from each of the word from a given review are added and normalized to the length of the entire review to ensure that all the vectors have constant variance. Following is the function that takes the data frame of reviews (X) and the word2vec (GloVe in this case) model as inputs and converts it to a vector. Also, since most of the predictors are categorical variables, I performed **one-hot encoding** to convert them to numeric values.

```
def word2vec(X,model):
    result_array=[]
    t1=time.time()
    for i in range(1,len(X)):
            x_strip=re.sub(r'[^\w\s]','',X[i]) #strip the punctuation
            list_token=x_strip.split()
            token_vec=np.zeros(100)

            for j in range(1,len(list_token)):
                if list_token[j] in model.vocab:
                    token_vec=token_vec+model.get_vector(list_token[j])
                else:
                    token_vec=token_vec
            token_vec=(token_vec.reshape(-1,1))/len(list_token)
            token_vec=token_vec.squeeze()
            result_array.append(token_vec)

    result_df=pd.DataFrame(result_array)
    t2=time.time()
    return(result_df)
```

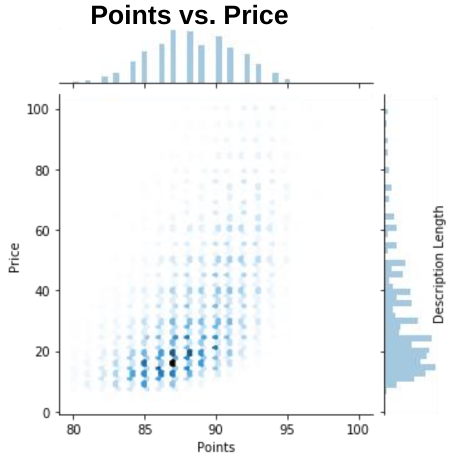
Descriptive Statistics

As for the exploratory data analysis, the first step would be to identify the underlying distribution of our dependent variable (points).

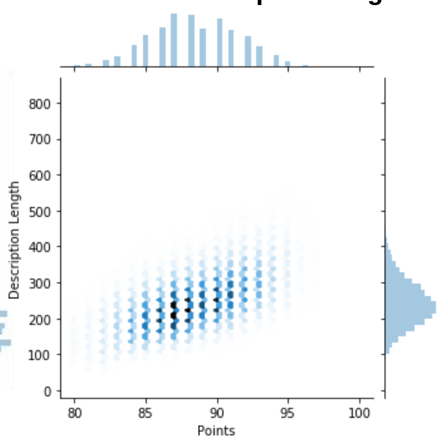
Points vs. No. of Wines



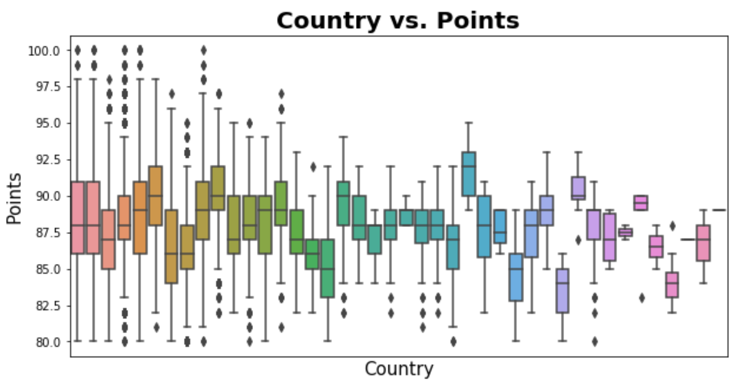
Points vs. Price

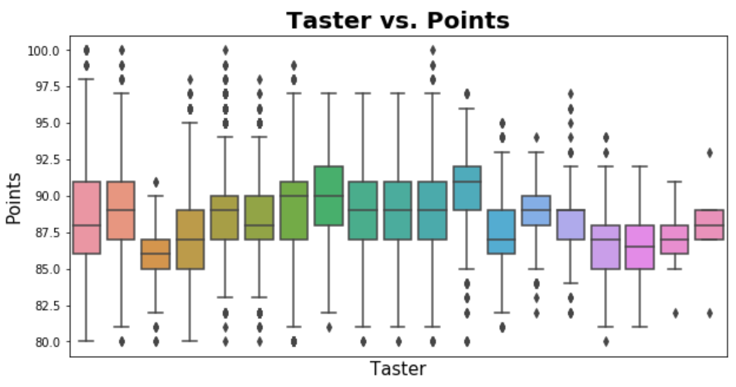


Points vs. Description length



The **points vs. No. of wines** plot suggests that the points follow an approximate normal distribution. Secondly, price usually has a strong influence on the quality of the wine. So will plot the wine points vs. price to see if there is a strong correlation. From the **points vs. price** plot, one can infer that there is a weak positive correlation between the points and the price. Hence including the price in our model would be good idea. Lastly from the points vs. description length plot, it seems like there is a very strong positive correlation between the length of the review and the points. Hence this could be used as an engineered feature to predict the points. Hence will be including all these predictors in the model.



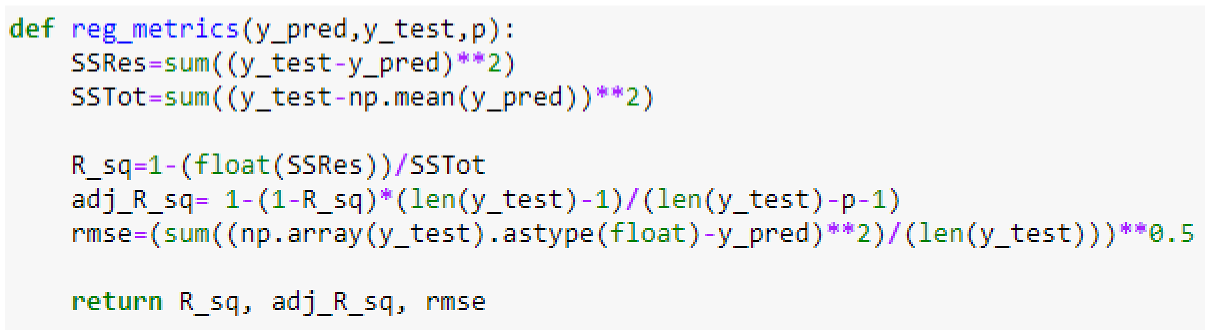


From the above two box plots, it seems that the **country of origin** and the **taster** who reviewed the wine do exhibit a significant variation. However, there is no obvious pattern and it will be interesting to see if the final model picks up any of these covariates. Since the taster's name and twitter handle essentially capture the same information, I am not including either of them.

Lastly, I will be excluding **region1**, **region2** and **title** for now. All these covariates have either too many NULL values or do not really contribute towards the points. **Winery** is another feature that could potentially be used, but since there are too many unique values for the winery category, I will be ignoring that too for now.

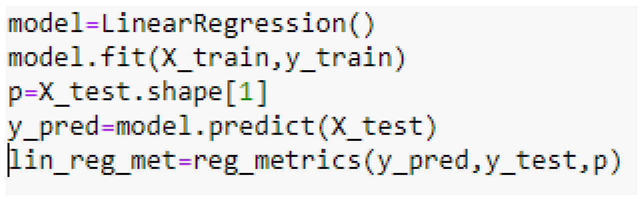
Regression Analysis

I will be evaluating the model's performance based on Adjusted R^2 and Root mean square error (RMSE). So defined a function to calculate these metrics for a given model.



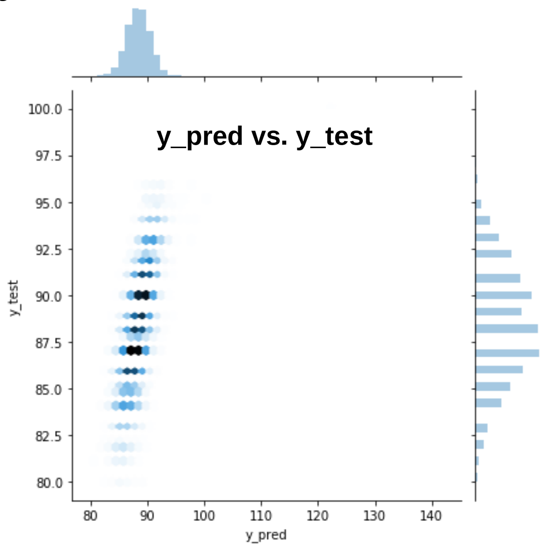
```
def reg_metrics(y_pred,y_test,p):
    SSRes=sum((y_test-y_pred)**2)
    SSTot=sum((y_test-np.mean(y_pred))**2)

    R_sq=1-(float(SSRes))/SSTot
    adj_R_sq= 1-(1-R_sq)*(len(y_test)-1)/(len(y_test)-p-1)
    rmse=(sum((np.array(y_test).astype(float)-y_pred)**2)/(len(y_test)))**0.5

    return R_sq, adj_R_sq, rmse
```

1) **Linear Regression:** Performed a multivariate linear regression on the data to have a baseline model for prediction.



```
model=LinearRegression()
model.fit(X_train,y_train)
p=X_test.shape[1]
y_pred=model.predict(X_test)
lin_reg_met=reg_metrics(y_pred,y_test,p)
```

```
Metrics for the MLR model-
R^2: 0.446 , Adj_R^2: 0.40 , RMSE 2.30
```

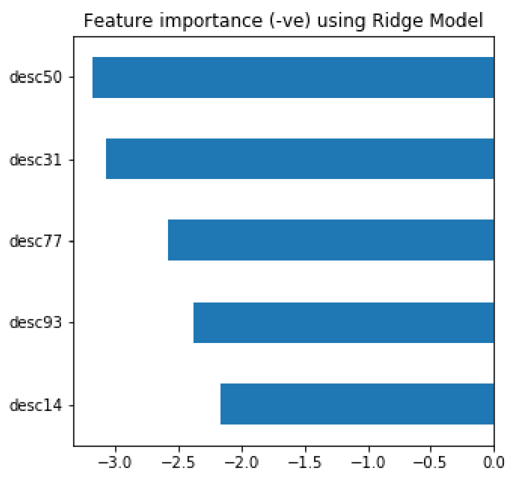


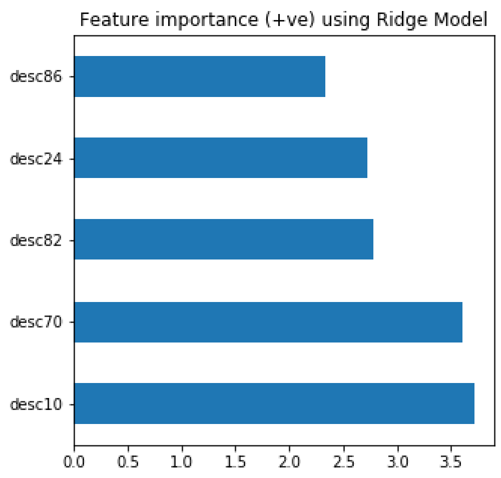
Based on the above metrics, the model can capture the variability in the data to some extent. However, from the relatively low adjusted R^2 value, we can conclude that there is more scope for improvement. This linear regression case will be considered as a base-line performance for further analysis. The y_pred vs. y_test graph also shows a reasonable correlation.

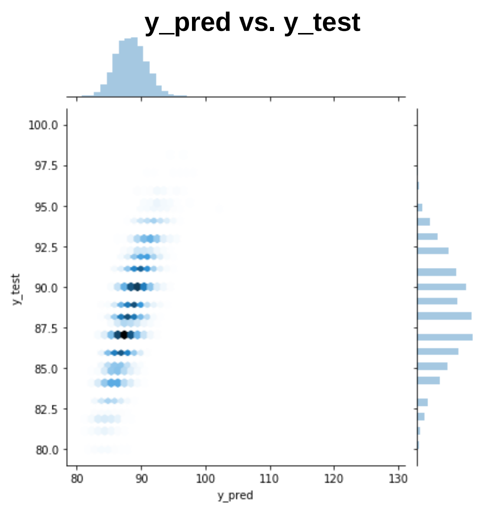
2) Ridge Regression:

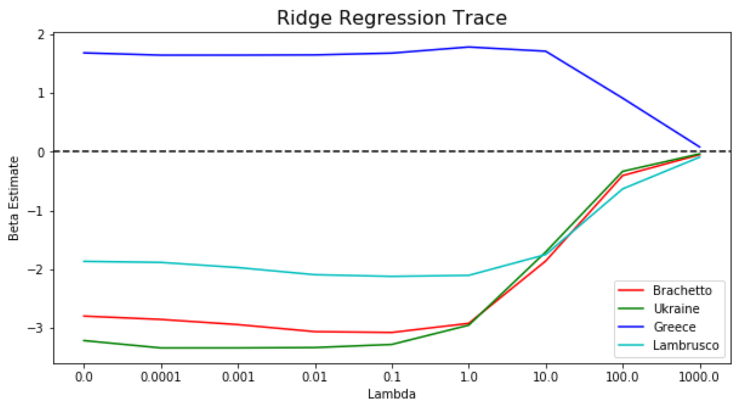
```python
alphas=np.array([0,0.0001,0.001,0.01, 0.1, 1, 10, 100, 1000])

#defining a custom scoring function to use RMSE
def mean_squared_error_(ground_truth, predictions):
    return mean_squared_error(ground_truth, predictions) ** 0.5
RMSE = make_scorer(mean_squared_error_, greater_is_better=False)
model=Ridge()
grid = GridSearchCV(estimator=model, param_grid=dict(alpha=alphas), scoring=RMSE, cv=5)
grid.fit(X_train,y_train)
```

Tried different values of alpha values for the ridge regression and tuned using a GridSearch with cross-validation. From this tuning, the optimal value of alpha turned out to be 10. Hence the final model is built using **α=10.**









Though the vectors corresponding to the description contributed the most towards predicting the wine review, there were a few other covariates that did matter. For example, the countries Greece/ Ukriane and the variety Brochette and Lambrusco did rank significantly high in their coefficient values (from the ridge regression trace). From the above Ridge trace, one can infer that the country Greece had a positive influence on the wine rating while the country Ukraine and the varieties Brochette and Lambrusco had a negative influence on the wine points. Moreover, the trace of Greece went to zero later than the other traces which shows that it has higher contribution towards the overall points of the wine.

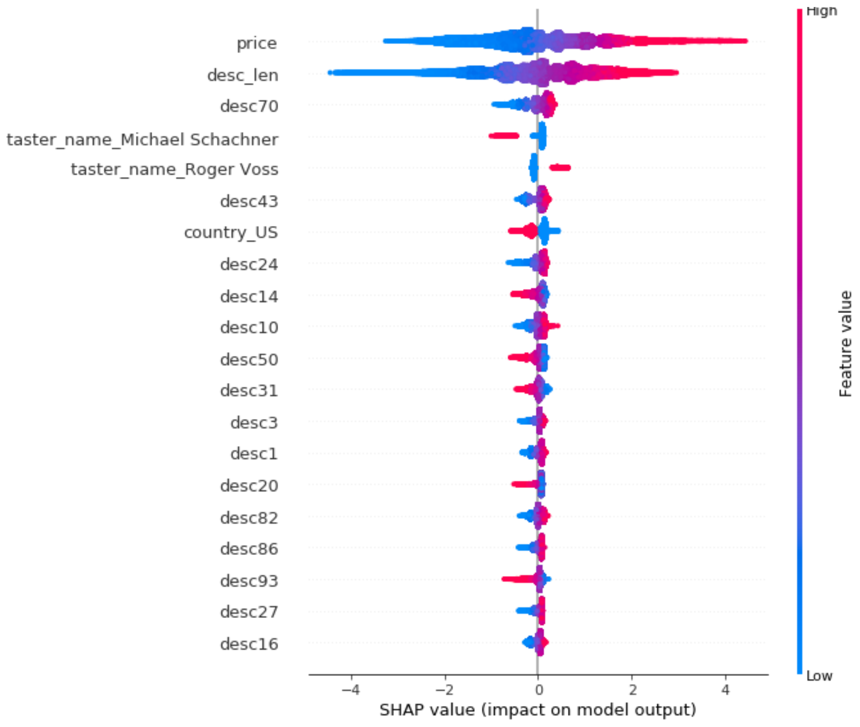
3) CatBoost

```
model=ctb.CatBoostRegressor(verbose=False)
parameters = {'depth'         : [3,5],
              'learning_rate' : [0.1],
              'iterations'    : [100,200,500]
             }
grid = GridSearchCV(estimator=model, param_grid = parameters, cv = 5, n_jobs=-1, scoring=RMSE)
grid.fit(X_train,y_train)
```

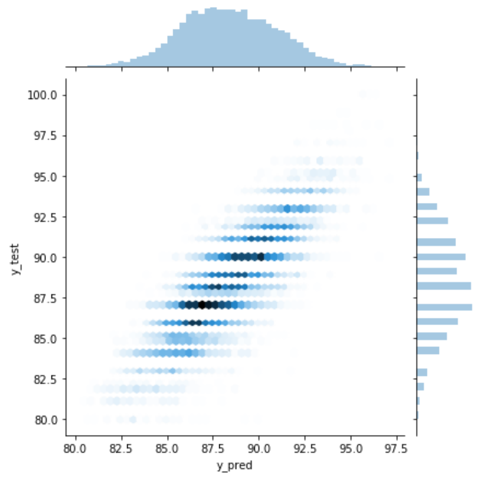
Performing the above grid search resulted in the following parameters being chosen:
`{'depth': 5, 'iterations': 500, 'learning_rate': 0.1}`
Using these parameters, the final model has been evaluated on the test data which led to the following evaluation metrics: `Metrics for the CatBoost model-`**`R^2:`**` 0.684, `**`Adj_R^2:`**` 0.658, `**`RMSE`**` 1.738`

To evaluate the performance of the **CatBoost** model and get a sense of which features are being influential, I used two different metrics here.

- The inbuilt **PredictionValuesChange** feature of the CatBoost algorithm which shows how much on average the prediction changes if the feature value changes and bigger the value of importance, bigger on average is the change in the predicted value. In addition to some of the description values, the prominent ones are **Taster, Price, length of description** and **Country seems to be the most influential factors.**
- The second metric used was the **SHAP (SHapley Additive exPlanations)** values which breaks a prediction value into contributions from each feature. It measures the impact of a feature on a single prediction value in comparison to the baseline prediction (which is the mean of the target value for training set). Using the SHAP values too, a very similar set of predictors have been identified with Price and description length having the highest impact.



```
PredictionValueChange:
taster_name_Sean P. Sullivan,
desc14,taster_name_Anne Krebie
hl MW, country_US, desc43,tast
er_name_Roger Voss, desc70, ta
ster_name_Michael Schachner, d
esc_len, price
```



y_pred vs. y_test

As inferred from the adjusted **R^2, RMSE** value and the **y_pred vs y_test** plot, the **CatBoost** outperforms the previously tried linear regression and Ridge regression by a large margin. Hence, for this problem of predicting wine ratings, **CatBoost** might be a very viable option.

Part 2

To capture more information about the wine, I am going to assume that the following description for a fruity Pinot Noir wine-"A medium-bodied Pinot Noir with fresh red fruit flavors of cherries, strawberries, blackberries and raspberry". Using this sentence, I then generate a vector from the GloVe algorithm (described previously). Then, I find the closest point to this description from the input dataset using cosine similarity(pre-processed to remove wines that are not pinot noir and above $20). Using a dissimilarity matrix constructed based on **Gower Distance** (which can handle mixed type of data containing both categorical and continuous predictors), I then rank the wines based on their similarity to the given description. Simultaneously, I also rank all the wines based on their points and average the points for each of the winery. Finally, I combined these two rankings (after normalization) and then choose the top five. Using this approach, I not only account for the best wine rating points, but other features such as the country of origin, the review given by the taster, price and variety (all these are accounted for while calculating the Gower Distance) hence giving an overall best recommendation.

```
proc_data_q2=proc_data_q2.drop(['designation', 'description',  'province',
       'region_1', 'region_2', 'taster_twitter_handle', 'title'], axis=1)

df_pinot=proc_data_q2.loc[proc_data_q2['variety'] == 'Pinot Noir']
df_pinot_price=df_pinot.loc[df_pinot['price']<=20].dropna()
df_pinot_price=df_pinot_price.reset_index(drop=True)
df_pinot_price=df_pinot_price.drop(columns=['variety'])
```

```
gr_dist=gower_distance(df_pinot_price)
```

```
dist_pinot=df_pinot_price.loc[:,'desc0':'desc99']
cos_sim=cosine_similarity(dist_pinot, token_vec_cust.reshape(1,-1))
```

Once the similarity is calculated, I next construct a DataFrame that has the list of all wines arranged in descending order based on the cosine similarity. That is the best matching wines are ranked high and the worst matching wines are ranked lower. Then I combine the rankings from both the cosine similarity and the ones by aggregating points to come up with a new ranking for all the wineries.

```
pinot_final_rank=pinot_rank_desc+pinot_rank_points
```

Wineries ranked by the current approach:

Winery	Rank Score
Stadlmann	0.254
Pike Road	0.272
Byron	0.282
Starmont	0.299
Scenic Valley Farms	0.314

Wineries ranked only by points

Winery	Rank Score
Stadlmann	0.0
Lynmar	0.038
Alta Maria	0.0769
Markowitsch	0.0769
Charles Heintz	0.0769

Hence from the above combined rankings, the top 5 wineries for someone looking for a fruity Pinot Noir under $20 would be **Stadlmann, Pike Road, Byron, Starmont** and **Scenic Valley Farms**. This list is different from what one would have recommended strictly based on the points. Hence the current approach not just look at the points, but considers other variables too such as the description, country of origin, price, taster and variety etc.