PREDICTING WINE RATINGS

Unit 3 Supervised Learning Capstone

Chase Walker

RESEARCH QUESTION

OBJECTIVE:

Our goal will be to find the best of the best wines using our data set. We want to predict which wines lie above the 95th percentile of wine scores.

Our target variable "wine points" only includes scores 80 and above because our data source, the wine enthusiast website, only includes reviews of wines that make the score of 80 and above.

BACKGROUND:

Our data set consists of wine reviews from different wine tasters that give an overall score and written description of each wine. The origins and type of wine are tracked in different variables in the data set.

DATA OVERVIEW

ORIGINS:

Our raw data was uploaded to Kaggle by user Zackthoutt. The data was scraped from WineEnthusiast during the week of November 22nd, 2017. Raw data includes almost 130,000 wine reviews.

The data set is on Kaggle and can be retrieved HERE.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 129971 entries, 0 to 129970
Data columns (total 14 columns):
Unnamed: 0               129971 non-null int64
country                  129908 non-null object
description              129971 non-null object
designation              92506 non-null object
points                   129971 non-null int64
price                    120975 non-null float64
province                 129908 non-null object
region_1                 108724 non-null object
region_2                 50511 non-null object
taster_name              103727 non-null object
taster_twitter_handle    98758 non-null object
title                    129971 non-null object
variety                  129970 non-null object
winery                   129971 non-null object
dtypes: float64(1), int64(2), object(11)
memory usage: 13.9+ MB
None
```

DATA OVERVIEW

FEATURES:

ID	- Rating ID
Country	- Country where the wine originates
Description	- Description of wine
Designation	- Specific Vineyard within Winery
Points	- Points given to wine by taster
Price	- Price of wine
Province	- Province or State within the Country of origin
Region 1	- Specific region within Province if applicable
Region 2	- Specific region within Region_1 if applicable
Taster Name	- Taster writing the review
Twitter Handle	- Taster's twitter handle if applicable
Title	- Title of the review
Variety	- Type of wine
Winery	- Winery where the wine originates from

NULL VALUES AND SHAPE

```
In [4]:  print(wr.isnull().sum())

Unnamed: 0                   0
country                     63
description                  0
designation              37465
points                       0
price                     8996
province                    63
region_1                 21247
region_2                 79460
taster_name              26244
taster_twitter_handle    31213
title                        0
variety                      1
winery                       0
dtype: int64
```

```
In [6]:  print(wr.shape)

(129971, 14)
```

DATA CLEANING

Province is a more precise location than country and has more unique values to work with. We will drop country in favor of province. We will also drop the ID column.

In [155]:

```
#print(wr.isnull().sum())
print('\nConfirming all 63 null values in country are also null values in province: {}.'.format(len(wr[(wr.country.isnull())
print('\nNumber of unique countries: {}.'.format(wr.country.nunique()))
print('Number of unique provinces: {}.'.format(wr.province.nunique()))
```

```
Confirming all 63 null values in country are also null values in province: 63.

Number of unique countries: 43.
Number of unique provinces: 425.
```

In [156]:

```
#Drop column 0. Column 0 is just a copy of the index.
#Drop where country and province is NULL
wr = wr.drop(columns = ['Unnamed: 0', 'country'])
wr = wr[~wr.province.isnull()]
```

In [157]:

```
twit = wr.groupby(['taster_name', 'taster_twitter_handle']).size().reset_index()
print(twit)
```

```
           taster_name taster_twitter_handle      0
0     Anne Krebiehl MW           @AnneInVino   3676
1    Christina Pickard       @winewchristina      6
2          Fiona Adams             @bkfiona     27
3         Jeff Jenssen        @worldwineguys    469
4           Jim Gordon       @gordone_cellars   4177
5        Joe Czerwinski               @JoeCz   5145
6         Kerin O’Keefe         @kerinokeefe  10776
7        Lauren Buzzeo            @laurbuzz   1832
8        Matt Kettmann         @mattkettmann   6332
9    Michael Schachner          @wineschach  15127
10      Mike DeSimone        @worldwineguys    502
11       Paul Gregutt           @paulgwine    9531
12         Roger Voss          @vossroger  25512
13    Sean P. Sullivan        @wawinereport   4966
14    Susan Kostrzewa        @suskostrzewa   1080
15     Virginie Boone              @vboone   9537
```

Each taster name has a one to one relationship with taster twitter handles so we can drop the twitter handle column. In other words, each taster is writing reviews under one and only one twitter account.

In [158]:

```
wr = wr.drop(columns = ['taster_twitter_handle'])
```

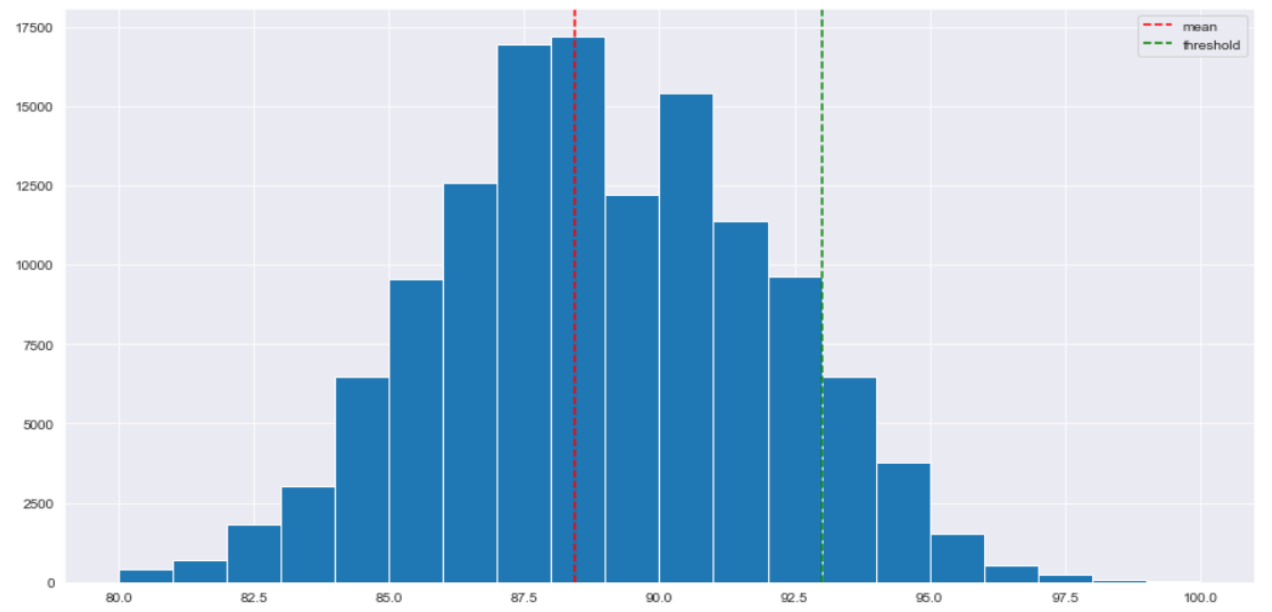
DATA CLEANING

Designation is the specific vineyard within the winery that the wine was made from. Although designation includes many null values, we can create a more specifc variable from winery + designation. Region_1 is the specific area within a province where the wine was made. Region_2 is the specific region within Region_1 where the wine was made. Region 2 has many null values and only 17 unique values so we can drop this column. Becuase Region_1 has 1229 unique values that can prove useful in determining points, we can combine province and region_1 to create a more specific variable with null values being 'Other'. We will fill in null values for taster name as 'Other' as well.

In [160]:
```
wr.designation = wr.designation.fillna(value = 'Other')
wr.region_1 = wr.region_1.fillna(value = 'Other')
wr.taster_name = wr.taster_name.fillna(value = 'Other')
wr = wr.drop(columns = ['region_2'])
```

Let's create a function that we will be using to evaluate the significance of ratios in our categorical variables. This will come in handy when trying to determine the relationship between a categorical variable and our target variable which is also categorical.

In [161]:
```
def plot_ratios(x, y, z):
    global ct
    ct = pd.crosstab(x, y, margins = True)
    ct = ct.div(ct.All, axis = 0)
    ct = ct.drop(columns = ['All'])
    ct = ct[:-1]
    mct = pd.melt(ct, id_vars = [z], value_vars = [0, 1])
    plt.figure(figsize = (15,8))
    sns.boxplot(mct.good_score, mct.value)
```




```
We will use the 95th percentile to represent a good score: 93
```

We will be using the median of the province the wine originates from to fill in Null values of price.

In [164]:
```
#wr.price = wr.price.fillna(value = 0)
#wr.province = wr.province.fillna(value = 'unknown')
temp = wr[['province', 'price']]
prov = temp.groupby(['province']).agg(np.median)
prov = prov.reset_index()
wr = wr.reset_index()
wr = wr.drop(columns = 'index')
```

In [165]:
```
for x in range(0,len(wr.price)):
    if math.isnan(wr.price[x]):
        if prov.province.isin([str(wr.province[x])]).any():
            for y in range(0,len(prov.province)):
                if prov.province[y] == wr.province[x]:
                    wr.price[x] = prov.price[y]
                else:
                    pass
        else:
            pass
    else:
        pass
```

String values can be marked as other if Null value is provided.

In [159]:
```
print('Null Count')
print(wr.isnull().sum())
print('\nUnique Values Count')
print(wr.nunique())
```

```
Null Count
description        0
designation    37454
points             0
price           8992
province           0
region_1       21184
region_2       79397
taster_name    26244
title              0
variety            1
winery             0
dtype: int64
```

We can drop the remaining 3 null values in price and variety that did not have a province to impute a price from.

In [166]:
```
wr = wr.dropna(subset = ['price', 'variety'])
```

In [167]:
```
print(wr.isnull().sum())
```

```
description    0
designation    0
points         0
price          0
province       0
region_1       0
taster_name    0
title          0
variety        0
winery         0
good_score     0
dtype: int64
```

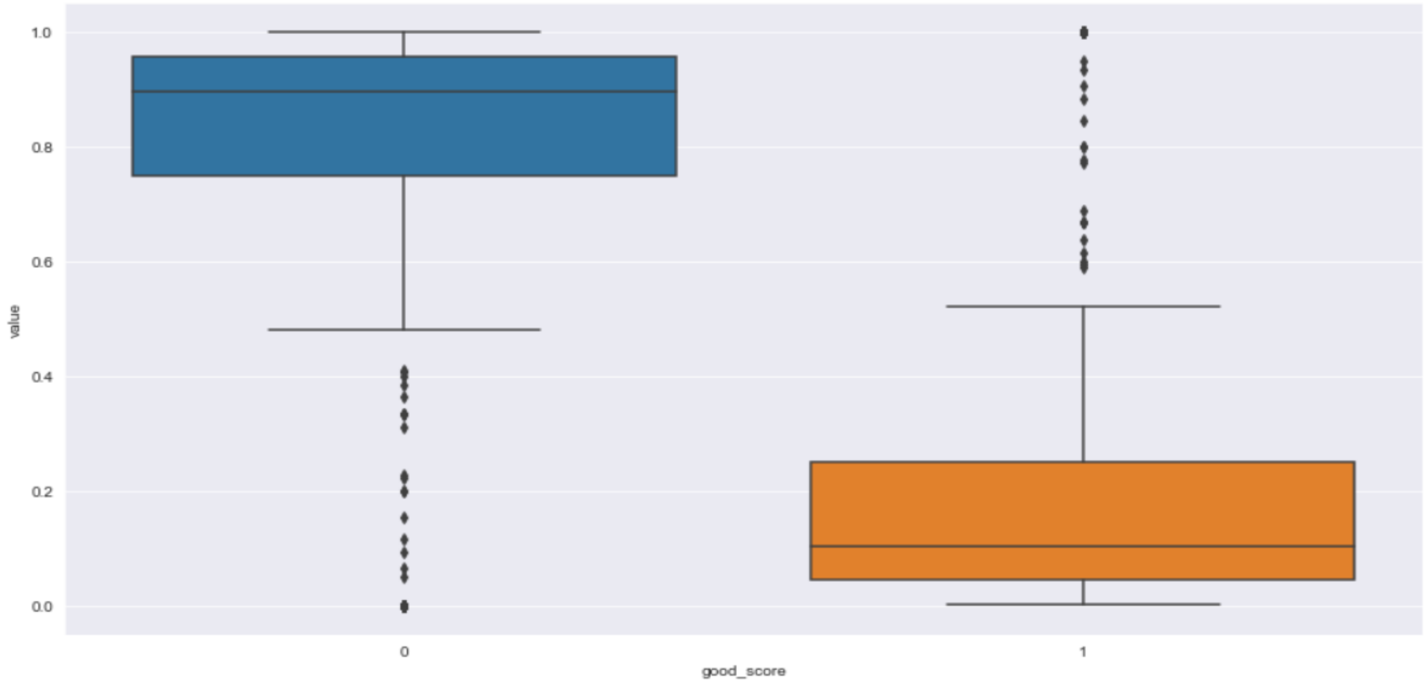
DATA EXPLORATION

Our dataset is looking a lot better. Let's do some EDA. First, we are going to combine the province and region columns to get a more accurate location of wine.

In [168]:
```
wr['province_region'] = wr.province + wr.region_1
wr = wr.drop(columns = ['province', 'region_1'])
```

Using our plot_ratios function, we can find the ratio of good wine to bad wine per province/region which will help us reduce some dimensionality of categorical variables. Our data set would contain too much noise if we used dummy variables for each province/region combination.

In [170]:
```
low_prov_reg = []
for x in range(0,len(ct[0])):
    if ct[0][x] == 1:
        low_prov_reg.append(ct.index[x])
    else:
        pass
wr['very_low_province_region'] = np.where(wr.province_region.isin(low_prov_reg), 1, 0)

#Let's regraph our countries without countries that only have bad reviews.
wr['province_region_2'] = np.where(wr.very_low_province_region == 1, 'Other', wr.province_region)
df_temp = wr[['province_region_2', 'good_score']]
df_temp = df_temp[df_temp['province_region_2'] != 'Other']

plot_ratios(df_temp.province_region_2, df_temp.good_score, 'low_prov_reg')
print(ct)
```



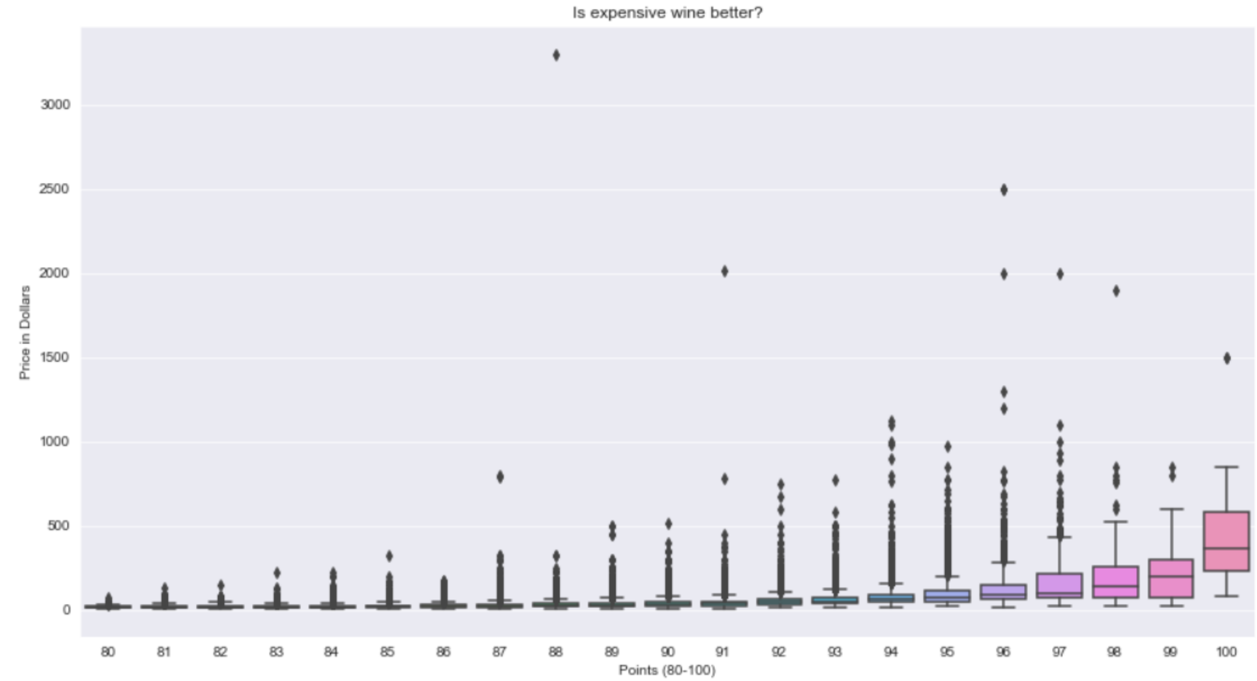
FEATURE ENGINEERING

We will split the province_region column into 6 separate binary classes. Very low will contain ratio percentages of 0. Low will contain percentages of good wine from 1 to 25 percent. Mid will contain wine percentages from 26 - 50, high will contain percentages from 51 - 75, and very high will contain percentages from 76 - 99. 100 will not need a separate column because we can deduce that if the variable is now low, mid, or high, the variable must be 100.

In [171]:
```
low_province_region = []
for x in range(0,len(ct[0])):
    if (ct[1][x] <= 0.25) & (ct[1][x] > 0):
        low_province_region.append(ct.index[x])
    else:
        pass
wr['low_province_region'] = np.where(wr.province_region.isin(low_province_region), 1, 0)

mid_province_region = []
for x in range(0,len(ct[0])):
    if (ct[1][x] <= 0.5) & (ct[1][x] > 0.25):
        mid_province_region.append(ct.index[x])
    else:
        pass
wr['mid_province_region'] = np.where(wr.province_region.isin(mid_province_region), 1, 0)

high_province_region = []
for x in range(0,len(ct[0])):
    if (ct[1][x] <= 0.75) & (ct[1][x] > 0.5):
        high_province_region.append(ct.index[x])
    else:
        pass
wr['high_province_region'] = np.where(wr.province_region.isin(high_province_region), 1, 0)

very_high_province_region = []
for x in range(0,len(ct[0])):
    if (ct[1][x] <= 1) & (ct[1][x] > 0.75):
        high_province_region.append(ct.index[x])
    else:
        pass
wr['very_high_province_region'] = np.where(wr.province_region.isin(very_high_province_region), 1, 0)
```

DATA EXPLORATION

In [22]:
```
plt.figure(figsize = (15,8))
sns.boxplot(wr.points, wr.price)
plt.title('Is expensive wine better?')
plt.xlabel('Points (80-100)')
plt.ylabel('Price in Dollars')
plt.show()
```



A couple take-aways from this graph:

1. Scores are given in integers.
2. Cheap wine can get really good scores.
3. Expensive wines tend to be scored higher.

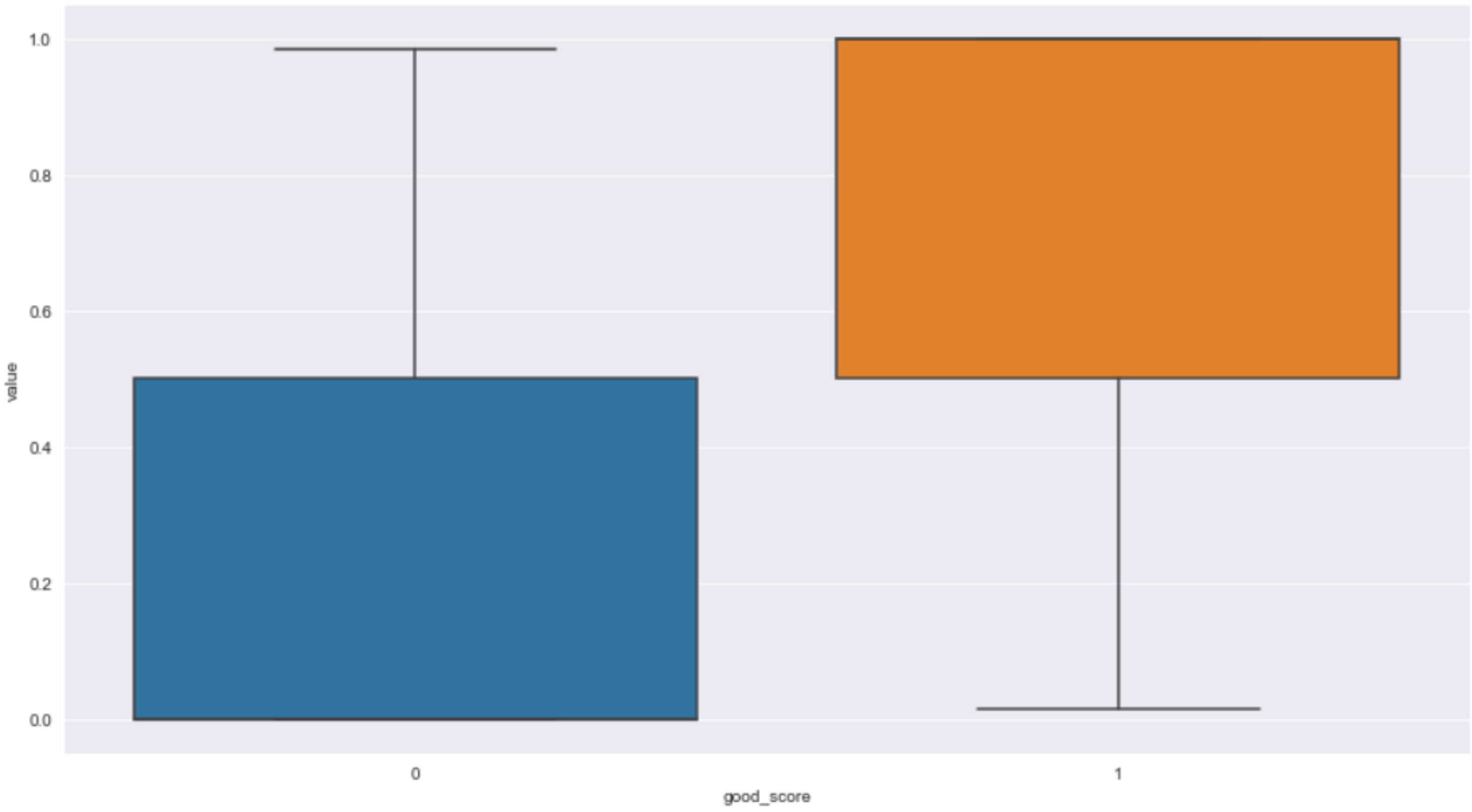
EX: Once our wine is $500 a bottle, we only have a few wines that are considered under our "93" score threshold.

DATA EXPLORATION

Let's combine our winery and designation columns.

In [172]:
```
wr['winery_designation'] = wr.winery + wr.designation
```

In [174]:
```
low_win_des = []
for x in range(0,len(ct[0])):
    if ct[0][x] == 1:
        low_win_des.append(ct.index[x])
    else:
        pass
wr['very_low_wine_designation'] = np.where(wr.winery_designation.isin(low_win_des), 1, 0)

#Let's regraph our countries without countries that only have bad reviews.
wr['winery_designation_2'] = np.where(wr.very_low_wine_designation == 1, 'Other', wr.winery_designation)
df_temp = wr[['winery_designation_2', 'good_score']]
df_temp = df_temp[df_temp['winery_designation_2'] != 'Other']

plot_ratios(df_temp.winery_designation_2, df_temp.good_score, 'winery_designation_2')
print(ct)
```



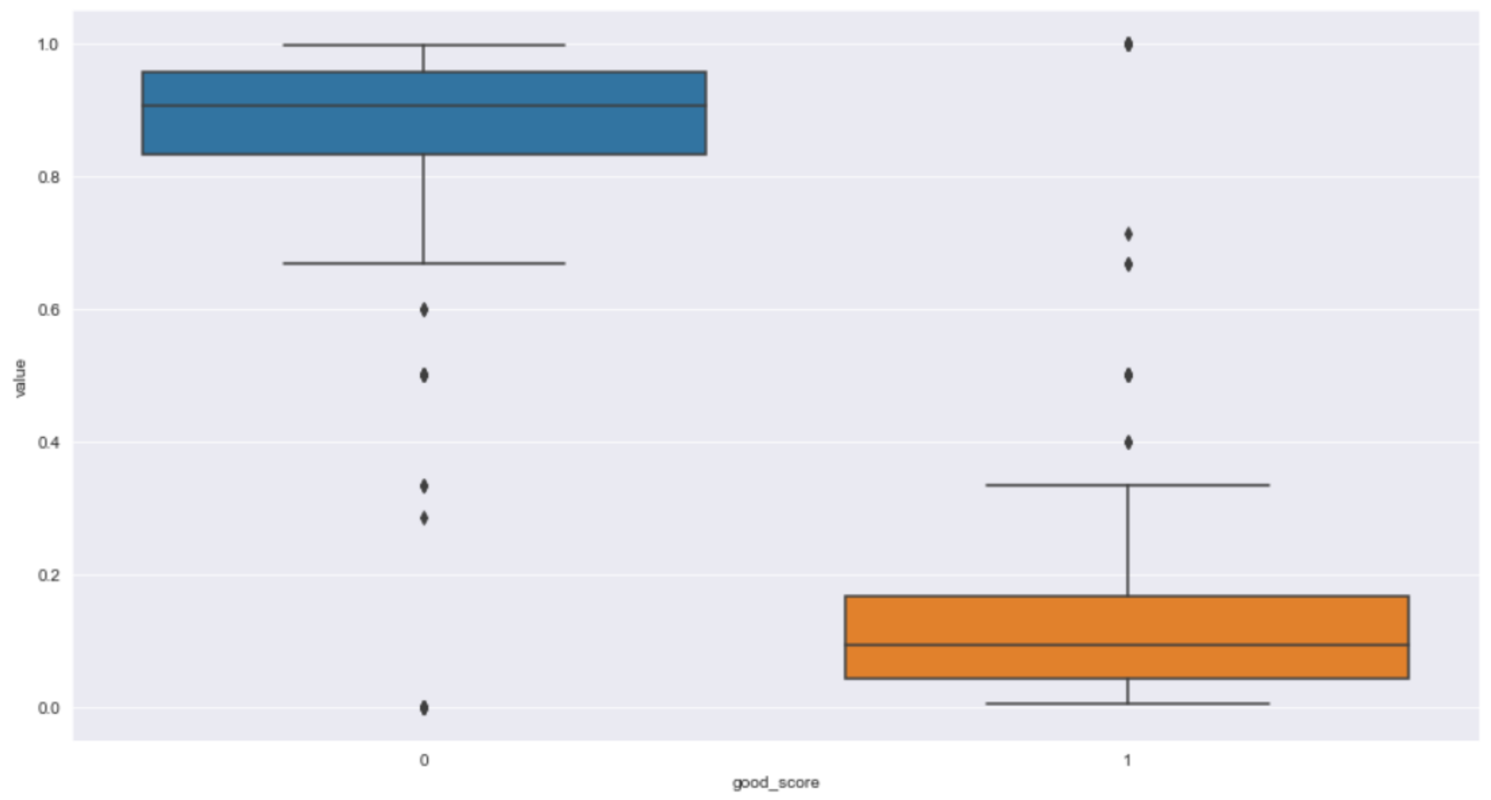
FEATURE ENGINEERING

In [175]:
```
low_winery_designation = []
for x in range(0,len(ct[0])):
    if (ct[1][x] <= 0.25) & (ct[1][x] > 0):
        low_winery_designation.append(ct.index[x])
    else:
        pass
wr['low_winery_designation'] = np.where(wr.winery_designation.isin(low_winery_designation), 1, 0)

mid_winery_designation = []
for x in range(0,len(ct[0])):
    if (ct[1][x] <= 0.5) & (ct[1][x] > 0.25):
        mid_winery_designation.append(ct.index[x])
    else:
        pass
wr['mid_winery_designation'] = np.where(wr.winery_designation.isin(mid_winery_designation), 1, 0)

high_winery_designation = []
for x in range(0,len(ct[0])):
    if (ct[1][x] <= 0.75) & (ct[1][x] > 0.5):
        high_winery_designation.append(ct.index[x])
    else:
        pass
wr['high_winery_designation'] = np.where(wr.winery_designation.isin(high_winery_designation), 1, 0)

very_high_winery_designation = []
for x in range(0,len(ct[0])):
    if (ct[1][x] <= 1) & (ct[1][x] > 0.75):
        very_high_winery_designation.append(ct.index[x])
    else:
        pass
wr['very_high_winery_designation'] = np.where(wr.winery_designation.isin(very_high_winery_designation), 1, 0)
```

DATA EXPLORATION

We will do the same process for variety.

In [177]:
```
low_var = []
for x in range(0,len(ct[0])):
    if ct[0][x] == 1:
        low_var.append(ct.index[x])
    else:
        pass
wr['very_low_variety'] = np.where(wr.variety.isin(low_var), 1, 0)

#Let's regraph our countries without countries that only have bad reviews.
wr['variety2'] = np.where(wr.very_low_variety == 1, 'Other', wr.variety)
df_temp = wr[['variety2', 'good_score']]
df_temp = df_temp[df_temp['variety2'] != 'Other']

plot_ratios(df_temp.variety2, df_temp.good_score, 'variety2')
print(ct)
```



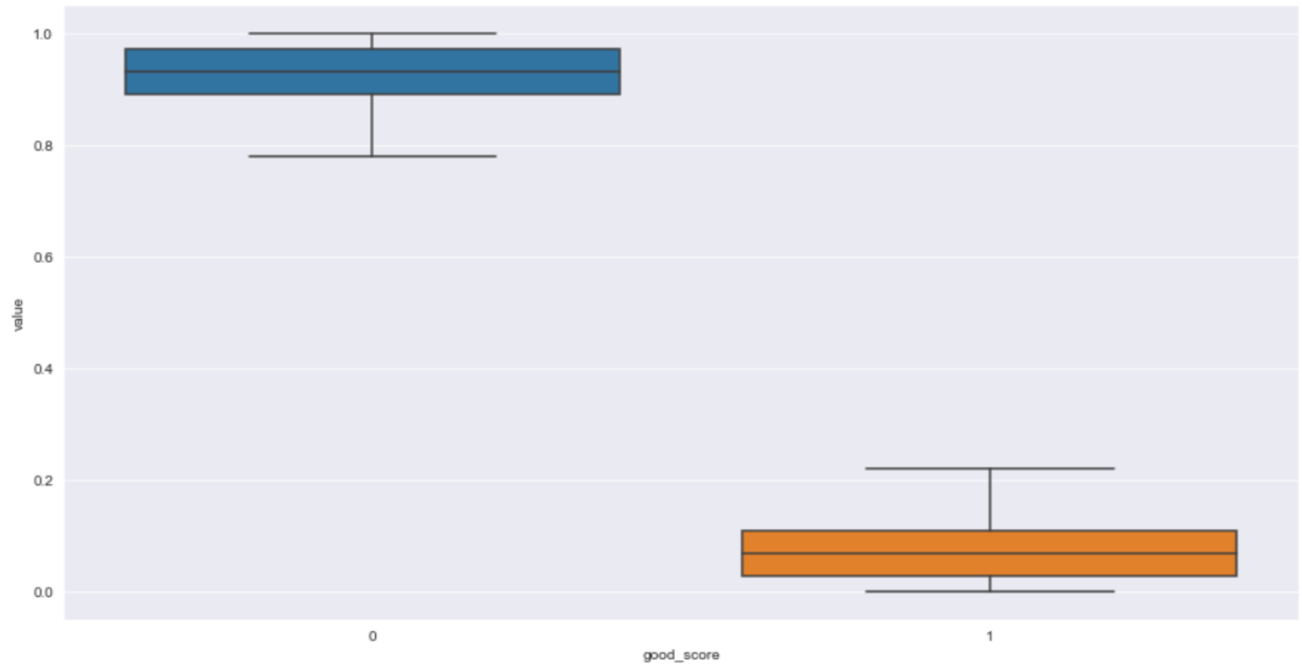
FEATURE ENGINEERING

In [178]:
```
low_variety = []
for x in range(0,len(ct[0])):
    if (ct[1][x] <= 0.25) & (ct[1][x] > 0):
        low_variety.append(ct.index[x])
    else:
        pass
wr['low_variety'] = np.where(wr.variety.isin(low_variety), 1, 0)

mid_variety = []
for x in range(0,len(ct[0])):
    if (ct[1][x] <= 0.5) & (ct[1][x] > 0.25):
        mid_variety.append(ct.index[x])
    else:
        pass
wr['mid_variety'] = np.where(wr.variety.isin(mid_variety), 1, 0)

high_variety = []
for x in range(0,len(ct[0])):
    if (ct[1][x] <= 0.75) & (ct[1][x] > 0.5):
        high_variety.append(ct.index[x])
    else:
        pass
wr['high_variety'] = np.where(wr.variety.isin(high_variety), 1, 0)

very_high_variety = []
for x in range(0,len(ct[0])):
    if (ct[1][x] <= 1) & (ct[1][x] > 0.75):
        very_high_variety.append(ct.index[x])
    else:
        pass
wr['very_high_variety'] = np.where(wr.variety.isin(very_high_variety), 1, 0)
```

DATA EXPLORATION

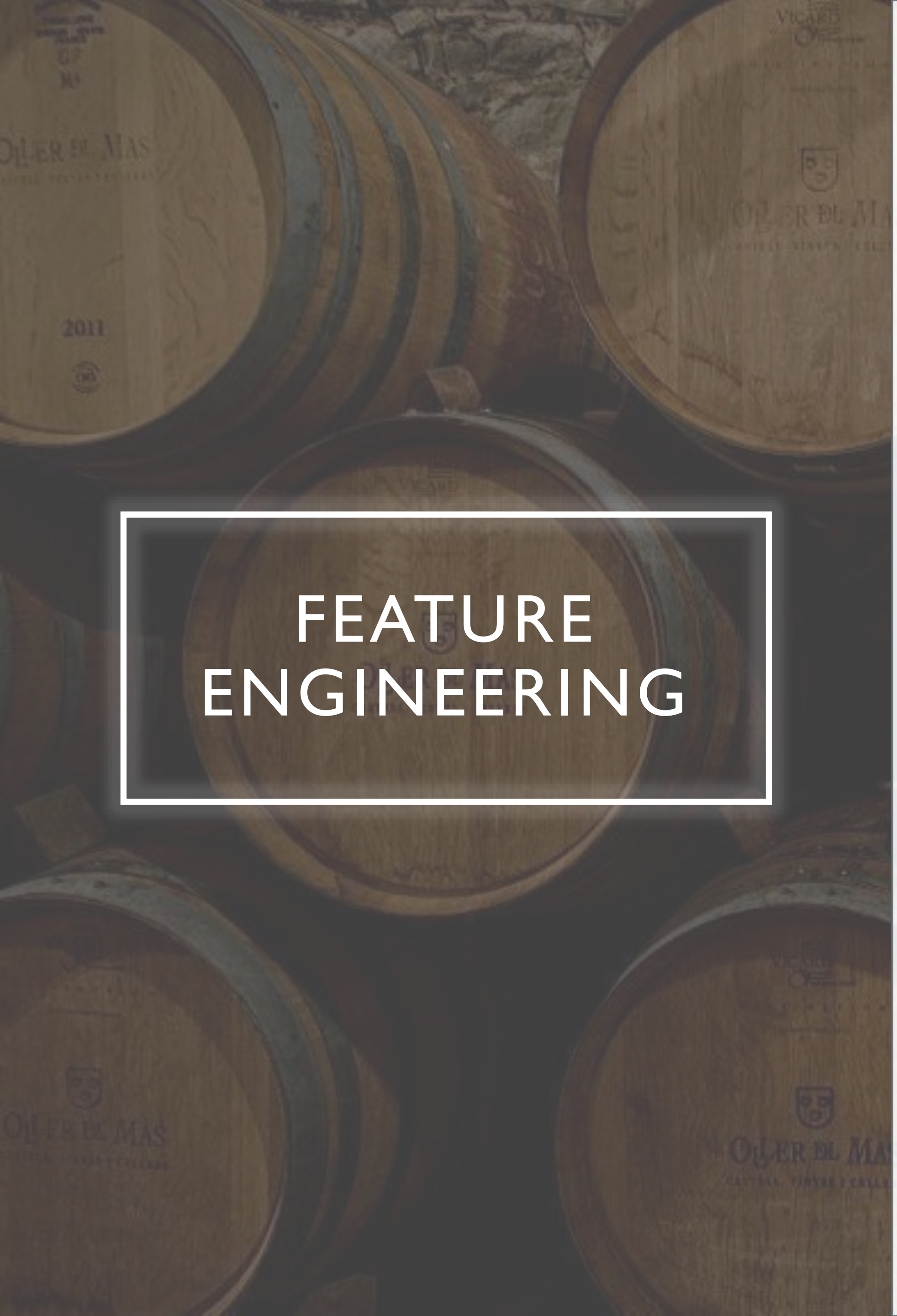
Let's investigate the spread of data for wine tasters.

In [179]:
```
plot_ratios(wr.taster_name, wr.good_score, 'taster_name')
```



In [180]:
```
print(ct)
```

```
good_score                0         1
taster_name
Alexander Peartree  1.000000  0.000000
Anna Lee C. Iijima  0.930691  0.069309
Anne Krebiehl MW    0.779652  0.220348
Carrie Dykes        1.000000  0.000000
Christina Pickard   0.833333  0.166667
Fiona Adams         1.000000  0.000000
Jeff Jenssen        0.976546  0.023454
Jim Gordon          0.930572  0.069428
Joe Czerwinski      0.929057  0.070943
Kerin O'Keefe       0.905809  0.094191
Lauren Buzzeo       0.971038  0.028962
Matt Kettmann       0.813329  0.186671
Michael Schachner   0.966550  0.033450
Mike DeSimone       0.970120  0.029880
Other               0.910490  0.089510
Paul Gregutt        0.894240  0.105760
Roger Voss          0.878175  0.121825
Sean P. Sullivan    0.940395  0.059605
Susan Kostrzewa     0.994439  0.005561
Virginie Boone      0.859285  0.140715
```



FEATURE ENGINEERING

```
In [181]: wr3 = pd.get_dummies(wr.taster_name)
          wr = pd.concat([wr, wr3], axis = 1)
          wr = wr.drop(columns = ['taster_name'])
```

```
In [182]: print(wr.columns)

          Index(['description', 'designation', 'points', 'price', 'title', 'variety',
                 'winery', 'good_score', 'province_region', 'very_low_province_region',
                 'province_region_2', 'low_province_region', 'mid_province_region',
                 'high_province_region', 'very_high_province_region',
                 'winery_designation', 'very_low_wine_designation',
                 'winery_designation_2', 'low_winery_designation',
                 'mid_winery_designation', 'high_winery_designation',
                 'very_high_winery_designation', 'very_low_variety', 'variety2',
                 'low_variety', 'mid_variety', 'high_variety', 'very_high_variety',
                 'Alexander Peartree', 'Anna Lee C. Iijima', 'Anne Krebiehl MW',
                 'Carrie Dykes', 'Christina Pickard', 'Fiona Adams', 'Jeff Jenssen',
                 'Jim Gordon', 'Joe Czerwinski', 'Kerin O’Keefe', 'Lauren Buzzeo',
                 'Matt Kettmann', 'Michael Schachner', 'Mike DeSimone', 'Other',
                 'Paul Gregutt', 'Roger Voss', 'Sean P. Sullivan', 'Susan Kostrzewa',
                 'Virginie Boone'],
                dtype='object')
```

```
In [183]: wr2 = wr.drop(columns = ['description',
                                   'designation',
                                   'points',
                                   'title',
                                   'variety',
                                   'winery',
                                   'province_region',
                                   'province_region_2',
                                   'winery_designation',
                                   'winery_designation_2',
                                   'variety2'])
```

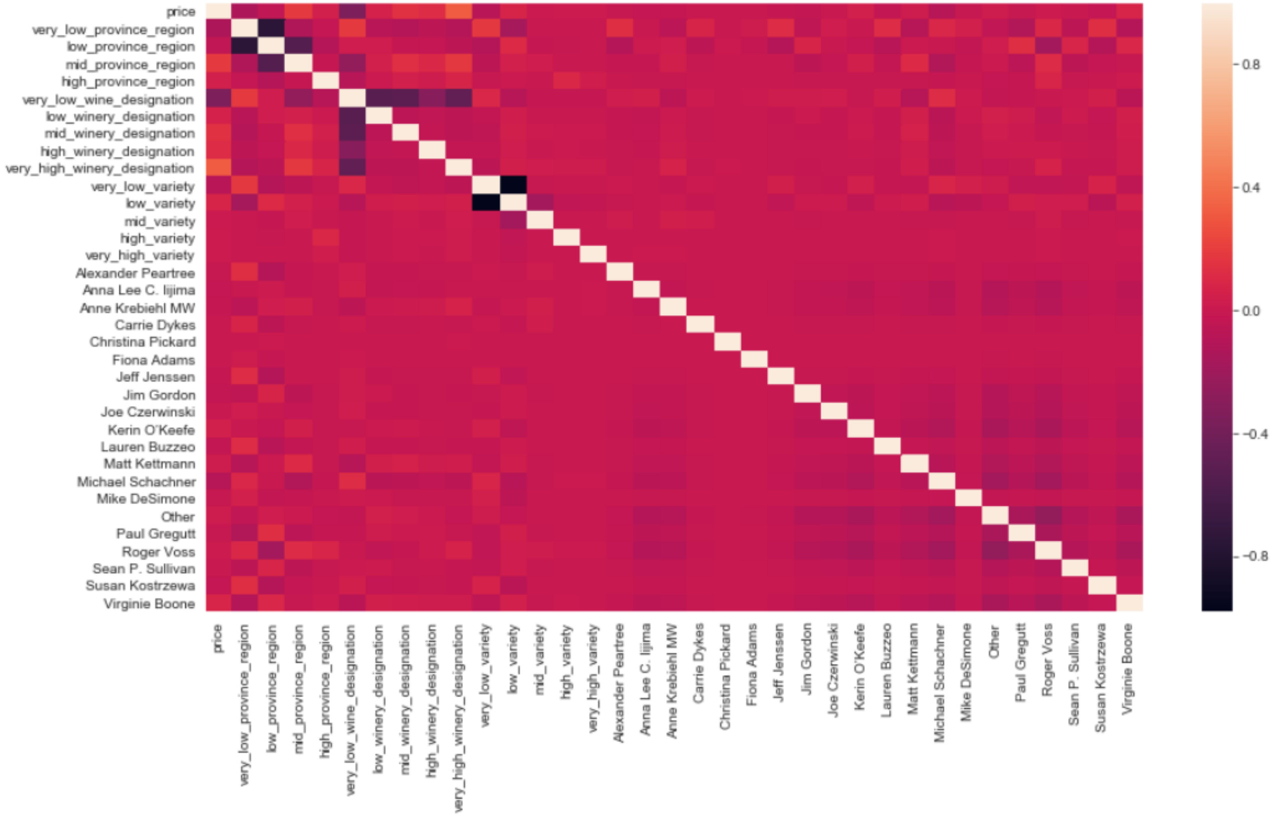
```
In [184]: print(wr2.columns)

          Index(['price', 'good_score', 'very_low_province_region',
                 'low_province_region', 'mid_province_region', 'high_province_region',
                 'very_high_province_region', 'very_low_wine_designation',
                 'low_winery_designation', 'mid_winery_designation',
                 'high_winery_designation', 'very_high_winery_designation',
                 'very_low_variety', 'low_variety', 'mid_variety', 'high_variety',
                 'very_high_variety', 'Alexander Peartree', 'Anna Lee C. Iijima',
                 'Anne Krebiehl MW', 'Carrie Dykes', 'Christina Pickard', 'Fiona Adams',
                 'Jeff Jenssen', 'Jim Gordon', 'Joe Czerwinski', 'Kerin O’Keefe',
                 'Lauren Buzzeo', 'Matt Kettmann', 'Michael Schachner', 'Mike DeSimone',
                 'Other', 'Paul Gregutt', 'Roger Voss', 'Sean P. Sullivan',
                 'Susan Kostrzewa', 'Virginie Boone'],
                dtype='object')
```

DATA EXPLORATION

```
In [186]: #Drop any features with all 0's in the column.
          wr2 = wr2.drop(columns = ['very_high_province_region'])
```

```
In [187]: correl = wr2.loc[:, ~wr2.columns.isin(['good_score'])]
          plt.figure(figsize = (15,8))
          sns.heatmap(correl.corr())
```

```
Out[187]: <matplotlib.axes._subplots.AxesSubplot at 0x2be994a7710>
```




```
In [50]: print(wr2.good_score.value_counts())
         print('\nGood Wines {}.'.format(12663/(117241+12663)))
         print('Bad Wines {}.'.format(117241/(117241+12663)))
```

```
0    117241
1     12663
Name: good_score, dtype: int64

Good Wines 0.09747967730016012.
Bad Wines 0.9025203226998398.
```

MODEL EVALUATION

LOGISTIC REGRESSION WITH CROSS VALIDATION

In [45]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score

lr = LogisticRegression()
X = wr2.loc[:, ~wr2.columns.isin(['good_score'])]
y = wr2.good_score

cvs = cross_val_score(lr, X, y, cv = 10)
print('Logistic Regression cross value scores: {}'.format(cvs))

print('Mean of scores is {}.'.format(np.mean(cvs)))
print('Std Dev of scores is {}.'.format(np.std(cvs)))
```

```
Logistic Regression cross value scores: [0.96005234 0.9592795  0.95966438 0.96212471 0.95673595 0.96243264
 0.96166282 0.95950731 0.96150885 0.95981524]
Mean of scores is 0.960278375299738.
Std Dev of scores is 0.0016206233128544594.
```

LOGISTIC REGRESSION WITH 20% HOLDOUT GROUP

In [46]:

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report

lr2 = LogisticRegression()
X2 = wr2.loc[:, ~wr2.columns.isin(['good_score'])]
y2 = wr2.good_score

X_train, X_test, y_train, y_test = train_test_split(X2, y2, test_size = 0.2)

print(lr2.fit(X_train, y_train).score(X_test, y_test))
print(lr2.fit(X2, y2).score(X2,y2))

y_pred = lr2.predict(X_test)

residuals = y_pred - y_test

print(confusion_matrix(y_test, y_pred))
print('\n')
print(classification_report(y_test, y_pred))
```

```
With 20% holdout: 0.9588545475539818
```

```
Without holdout group: 0.96045541322823
```

```
[[23165   312]
 [  752  1752]]
```

	precision	recall	f1-score	support
0	0.97	0.99	0.98	23477
1	0.85	0.70	0.77	2504
micro avg	0.96	0.96	0.96	25981
macro avg	0.91	0.84	0.87	25981
weighted avg	0.96	0.96	0.96	25981

MODEL EVALUATION

GRADIENT BOOSTING CLASSIFIER

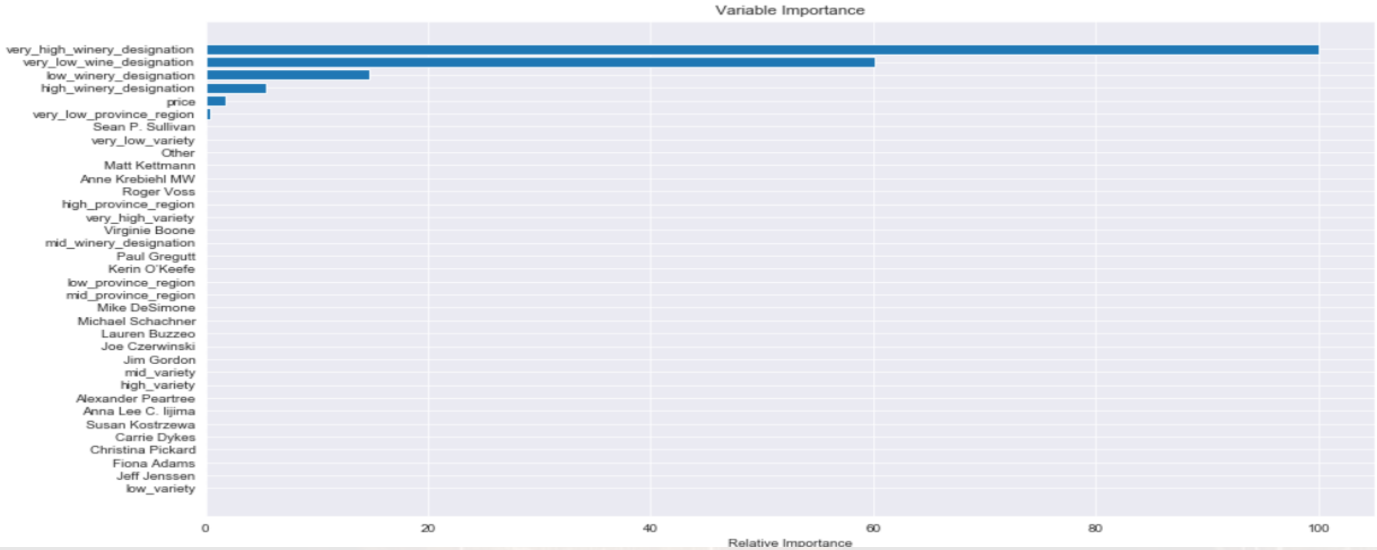
In [47]:

```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split, cross_val_score

Xgbc = wr2.loc[:, ~wr2.columns.isin(['good_score'])]
ygbc = wr2.good_score

gbc = GradientBoostingClassifier(n_estimators = 100, max_depth = 2, loss = 'deviance')

x_train, x_test, y_train, y_test = train_test_split(Xgbc, ygbc, test_size = 0.2)

print('With 20 percent holdout: {}.'.format(gbc.fit(x_train, y_train).score(x_test, y_test)))
print('Without holdout group: {}.'.format(gbc.fit(Xgbc, ygbc).score(Xgbc, ygbc)))

y_pred2 = gbc.predict(x_test)

acc2 = accuracy_score(y_test, y_pred2)
print('Accuracy is: {}.'.format(acc2))
```

```
With 20 percent holdout: 0.9588160578884569.
Without holdout group: 0.9604015272816849.
Accuracy is: 0.9593164235402795.
```

In [48]:

```
feature_importance = gbc.feature_importances_

# Make importances relative to max importance.
feature_importance = 100.0 * (feature_importance / feature_importance.max())
sorted_idx = np.argsort(feature_importance)
pos = np.arange(sorted_idx.shape[0]) + .5
plt.figure(figsize = (15,8))
plt.barh(pos, feature_importance[sorted_idx], align='center')
plt.yticks(pos, Xgbc.columns[sorted_idx])
plt.xlabel('Relative Importance')
plt.title('Variable Importance')
plt.show()
```

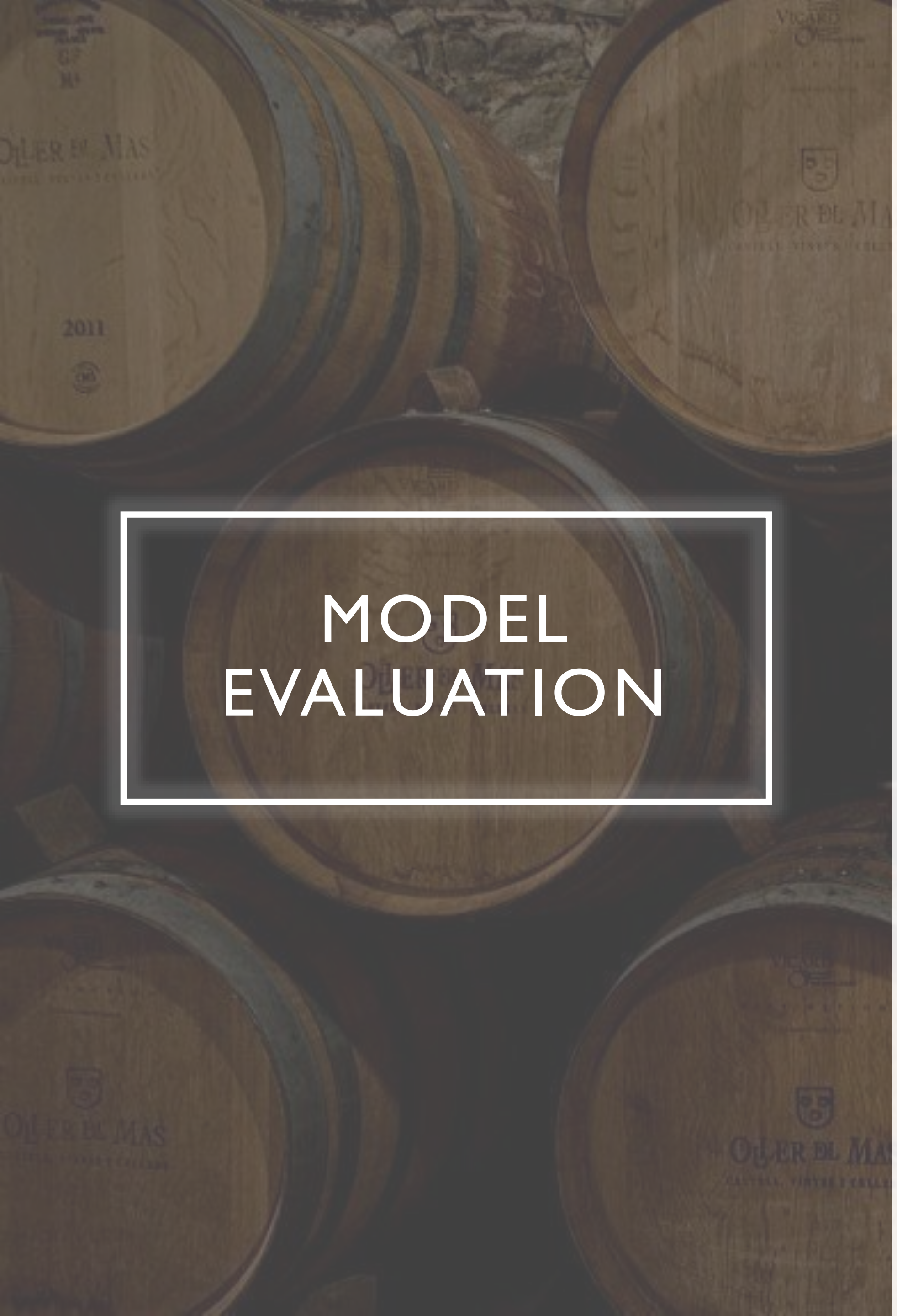


MODEL EVALUATION

SUPPORT VECTOR CLASSIFIER

```
In [42]: from sklearn.svm import SVC

         svc = SVC()
         Xsvc = wr2.loc[:, ~wr2.columns.isin(['good_score'])]
         ysvc = wr2.good_score

         cvs = cross_val_score(svc, Xsvc, ysvc, cv = 3)
         print('Support Vector Classification cross value scores: {}'.format(cvs))

         print('Mean of scores is {}.'.format(np.mean(cvs)))
         print('Std Dev of scores is {}.'.format(np.std(cvs)))
```

```
Support Vector Classification cross value scores: [0.95937832 0.95956213 0.95787626]
Mean of scores is 0.9589389053499943.
Std Dev of scores is 0.0007551405759199323.
```



NEXT STEPS

Although our best consistency comes from our SVC model, the model is very computationally intensive.

I will use the Gradient Boosting Classifier model to further evaluate and refine our features. Below is a data frame that extracts all features with 0% importance to our GBC model.

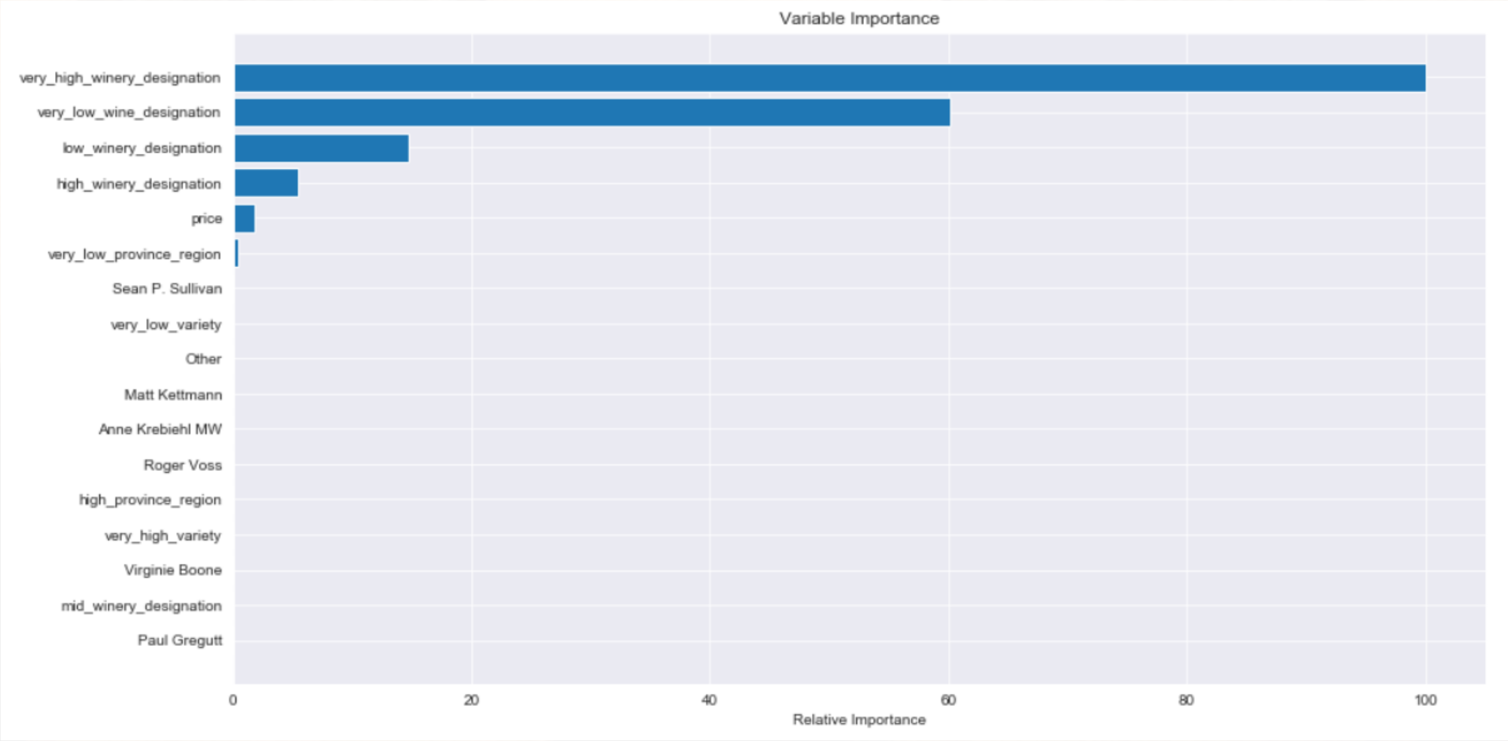
```
In [61]:  fi = pd.DataFrame()

          fi['col'] = Xgbc.columns[sorted_idx]
          fi['imp'] = np.sort(feature_importance)

          print(list(fi.col[fi.imp == 0]))

['low_variety', 'Jeff Jenssen', 'Fiona Adams', 'Christina Pickard', 'Carrie Dykes', 'Susan Kostrzewa', 'Anna Lee C. Iijima',
'Alexander Peartree', 'high_variety', 'mid_variety', 'Jim Gordon', 'Joe Czerwinski', 'Lauren Buzzeo', 'Michael Schachner',
'Mike DeSimone', 'mid_province_region', 'low_province_region', 'Kerin O'Keefe']
```

Our R^2 score remains relatively stable when significantly reducing our features.

```
In [62]:  wr3 = wr2.drop(columns = list(fi.col[fi.imp == 0]))

In [64]:  Xgbc2 = wr3.loc[:, ~wr3.columns.isin(['good_score'])]
          ygbc2 = wr3.good_score

          gbc = GradientBoostingClassifier(n_estimators = 100, max_depth = 2, loss = 'deviance')

          x_train, x_test, y_train, y_test = train_test_split(Xgbc2, ygbc2, test_size = 0.2)

          print('With 20 percent holdout: {}.'.format(gbc.fit(x_train, y_train).score(x_test, y_test)))
          print('Without holdout group: {}.'.format(gbc.fit(Xgbc2, ygbc2).score(Xgbc2, ygbc2)))

          y_pred3 = gbc.predict(x_test)

          acc3 = accuracy_score(y_test, y_pred3)
          print('Accuracy is: {}.'.format(acc3))

          With 20 percent holdout: 0.9586620992263577.
          Without holdout group: 0.9604015272816849.
          Accuracy is: 0.9589315268850314.
```

MODEL EVALUATION

NEXT STEPS

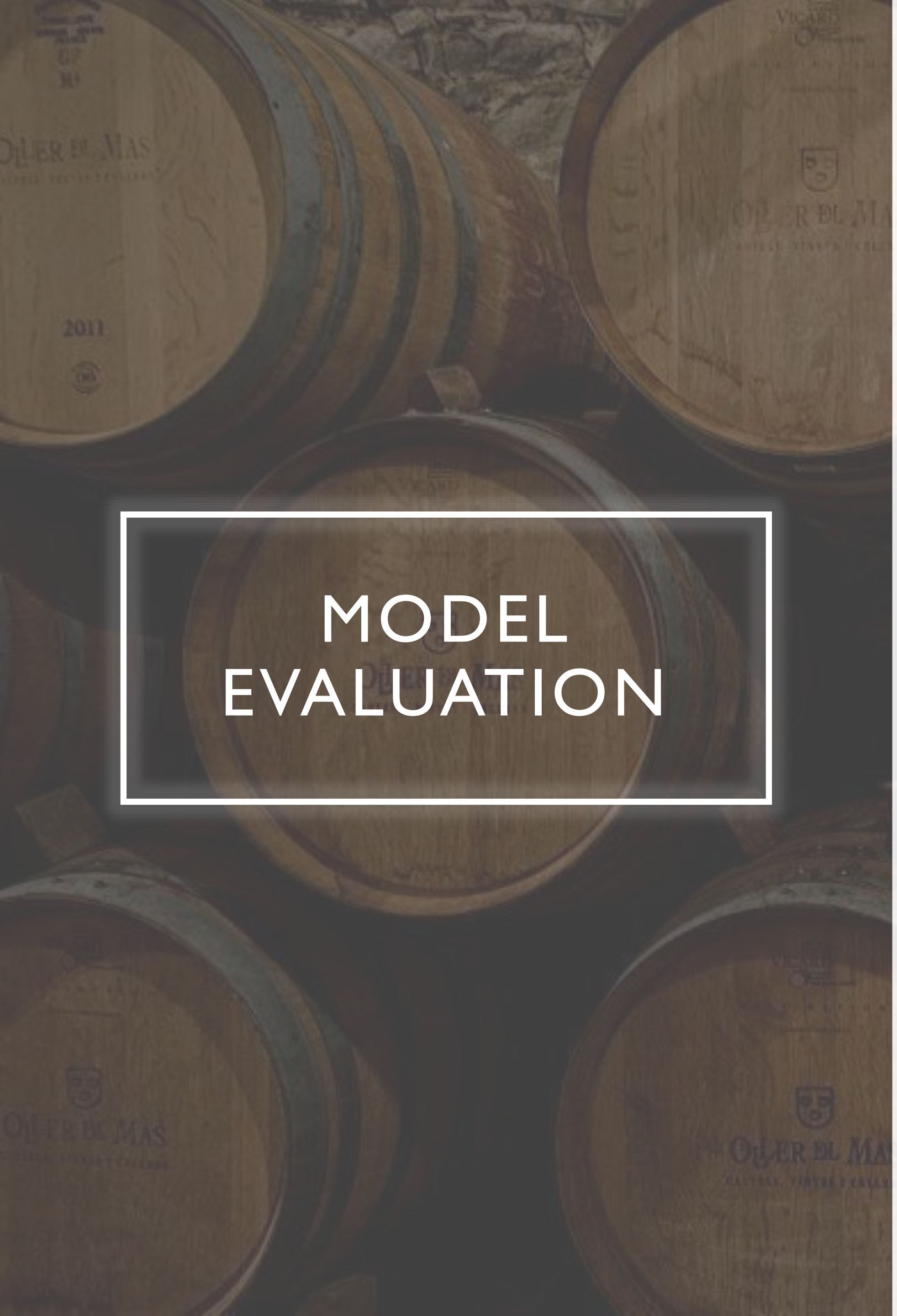


We can see from the below data frame that we no longer have any features with 0% importance. We could most likely reduce the features that are very close to 0% in our model at a small expense of accuracy, but because our model is already pretty simple, we will leave those variables in there.

In [66]:

```
fi = pd.DataFrame()

fi['col'] = Xgbc2.columns[sorted_idx]
fi['imp'] = np.sort(feature_importance)

print(list(fi.col[fi.imp == 0]))
```

```
[]
```



MODEL EVALUATION

FINAL NOTES ON EVALUATION

WITH EXTRA FEATURES

```
With 20 percent holdout: 0.9602786651783995.
Without holdout group: 0.9604015272816849.
Accuracy is: 0.9611639274854702.
[[23243   248]
 [  761  1729]]

              precision    recall  f1-score   support

           0       0.97      0.99      0.98     23491
           1       0.87      0.69      0.77      2490

   micro avg       0.96      0.96      0.96     25981
   macro avg       0.92      0.84      0.88     25981
weighted avg       0.96      0.96      0.96     25981
```

WITHOUT EXTRA FEATURES

```
With 20 percent holdout: 0.9581617335745353.
Without holdout group: 0.9604015272816849.
Accuracy is: 0.958392671567684.
[[23152   256]
 [  825  1748]]

              precision    recall  f1-score   support

           0       0.97      0.99      0.98     23408
           1       0.87      0.68      0.76      2573

   micro avg       0.96      0.96      0.96     25981
   macro avg       0.92      0.83      0.87     25981
weighted avg       0.96      0.96      0.96     25981
```

As we can see, the extra feature data set performed slightly better with less false readings on wine especially false negatives.

Precision was around the same value because our positive (true and false) were similar in both models.

Our recall and f1-score fell slightly due to the increase in false negatives in our simpler model.

MODEL EVALUATION

FINAL NOTES ON EVALUATION

```
In [66]:
from sklearn.model_selection import GridSearchCV

est_range = range(50,200,50)
dep_range = range(2,6)

param_grid = dict(n_estimators = est_range,
                  max_depth = dep_range,)

gbc = GradientBoostingClassifier(loss = 'deviance')
Xgbc3 = wr3.loc[:, ~wr3.columns.isin(['good_score'])]
ygbc3 = wr3.good_score

grid = GridSearchCV(gbc, param_grid, cv = 5, scoring = 'accuracy')

grid.fit(Xgbc3, ygbc3)
```

```
In [67]:
print(grid.best_score_)
print(grid.best_params_)
print(grid.best_estimator_)
```

```
0.9605400911442296
{'max_depth': 3, 'n_estimators': 150}
GradientBoostingClassifier(criterion='friedman_mse', init=None,
              learning_rate=0.1, loss='deviance', max_depth=3,
              max_features=None, max_leaf_nodes=None,
              min_impurity_decrease=0.0, min_impurity_split=None,
              min_samples_leaf=1, min_samples_split=2,
              min_weight_fraction_leaf=0.0, n_estimators=150,
              n_iter_no_change=None, presort='auto', random_state=None,
              subsample=1.0, tol=0.0001, validation_fraction=0.1,
              verbose=0, warm_start=False)
```

RESULTS

BEST SCORE: 96
BEST MAX DEPTH: 3
BEST N_ESTIMATORS: 150

MODEL EVALUATION

CONCLUSION

Our scores for the model are around 96% and our model has a relatively small amount of simple and powerful features.

DRAWBACKS

- The major drawback in our model is that only wine with 80 points or higher are featured on the website. Sentiment analysis becomes less accurate when distinguishing very good from great rather than good from bad. A major benefit to the data set is having written reviews and titles for each record which does not help much if we are evaluating between a B+ wine and an A- wine.
- Another flaw in the model is the winery and designation have a very strong influence on the rating of a wine. This means that if a wine originates from a vineyard with historically pretty good wine rather than excellent wine, the model will most likely evaluate the next wine to be below the threshold. Wine ratings like many food and drink ratings are subjective and very dependent on season, taster, preference, marketing and management. Without taking potentially influential variables into the model, we rely too much on just overall past performance of a particular vineyard.


THANK YOU