Preprocessing

In [34]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import cross_val_score, train_test_split, GridSearc
from sklearn.impute import SimpleImputer
from sklearn.ensemble import BaggingClassifier
from sklearn.metrics import plot_confusion_matrix, classification_report, f1_sco
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from sklearn.tree import DecisionTreeRegressor, DecisionTreeClassifier
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
from imblearn.over_sampling import SMOTE
from imblearn.pipeline import make_pipeline
from collections import Counter
from xgboost import XGBClassifier
import warnings
warnings.filterwarnings('ignore')

df = pd.read_csv('../../../data/EDA.csv')
```

Clarifying important predictor values (x) for the target value (y).

In [35]:
```
# dropping the target (y) values from x and specific origin
features = df.drop(['ref', 'company_manufacturer', 'company_location', 'review_d

X = features
y = df.rating_class
```

Splitting the data into three subsets of training and validation data for the future models.

```
Two train test splits create three subsets of the original dataset
which allows for the training data to not be bled into the test
data - this reduced model's bias towards the pre-existing testing
data, thus assuring maximum performance on future test sets to
which the model has never been exposed.
```

In [36]:
```
#performing train test split for test set (subsets 1/3)
X_tr, X_test, y_tr, y_test = train_test_split(X, y, test_size=.15, random_state=


#performing a train test split for train and validation set (subsets - 3/3)
X_train, X_val, y_train, y_val = train_test_split(X_tr, y_tr, test_size=.15, ran
```

In [37]:
```
#checking to ensure the shape of the columns and rows are still the same for the
X_tr.shape, y_tr.shape
```

Out[37]:
```
((1896, 8), (1896,))
```

Replacing any existing missing values

In [38]:
```
#ingredients have 88 rows that have no imputs.
X_val.isna().sum()
```

Out[38]:
```
memorable_characteristics_list    0
company_loc_bins                  0
bean_origin_bins                  0
weighted_specific_origin_bins     0
ingredient_list                   0
review_date_bin                   0
cocoa_bucket                      0
comp_manufact_bin                 0
dtype: int64
```

In [39]:
```
# the empty value replacement will be done using the most frequent fill strategy
imputer = SimpleImputer(missing_values=np.nan, strategy='most_frequent')

X_train_imputed = pd.DataFrame(imputer.fit_transform(X_train), columns = X.colum

X_val_imputed = pd.DataFrame(imputer.fit_transform(X_val),columns = X.columns)
```

In [40]:
```
#ensuring that the missing values from ingredients are now filled in with the mo

X_val_imputed.isna().sum()
```

Out[40]:
```
memorable_characteristics_list    0
company_loc_bins                  0
bean_origin_bins                  0
weighted_specific_origin_bins     0
ingredient_list                   0
review_date_bin                   0
cocoa_bucket                      0
comp_manufact_bin                 0
dtype: int64
```

Separating groups into numeric and catagorical data types

In [41]:
```
#taking a look at the groups data types to ensure that they are separated correc
X_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1611 entries, 1258 to 1818
Data columns (total 8 columns):
 #   Column                          Non-Null Count  Dtype 
---  ------                          --------------  ----- 
 0   memorable_characteristics_list  1611 non-null   object
 1   company_loc_bins                1611 non-null   int64 
 2   bean_origin_bins                1611 non-null   int64 
 3   weighted_specific_origin_bins   1611 non-null   int64 
 4   ingredient_list                 1611 non-null   object
 5   review_date_bin                 1611 non-null   object
 6   cocoa_bucket                    1611 non-null   object
 7   comp_manufact_bin               1611 non-null   object
dtypes: int64(3), object(5)
memory usage: 113.3+ KB
```

One Hot Encode Categorical Features

In [44]:
```
ohe = OneHotEncoder(sparse=False, handle_unknown='ignore')
```

In [45]:
```
# fit on training categorical data
ohe.fit(X_train)
X_train_encoded = ohe.transform(X_train)
X_val_encoded = ohe.transform(X_val)
X_train_encoded_df = pd.DataFrame(X_train_encoded, columns=ohe.get_feature_names
X_val_encoded_df= pd.DataFrame(X_val_encoded, columns=ohe.get_feature_names())
```

Modeling:

Model 1: Logistic Regression

In [46]:
```
# Instantiating a Logistic Regression model
logreg1 = LogisticRegression(class_weight={0: 0.77, 1: 0.23},
                             solver='newton-cg')
```

In [47]:
```
def modeling_function(model, X_train, y_train, X_val, y_val):

    # fit model on training data
    model.fit(X_train, y_train)

    # make predictions on training and validation data
    train_preds = model.predict(X_train)
    val_preds = model.predict(X_val)

    # Print accuracy score
    print('Training score: ', f1_score(y_train, train_preds))
    print('Validation score: ', f1_score(y_val, val_preds))

    # return fitted model
    return model
```

In [48]:
```
# call modeling function
logreg_1 = modeling_function(logreg1, X_train_encoded_df, y_train, X_val_encoded
```

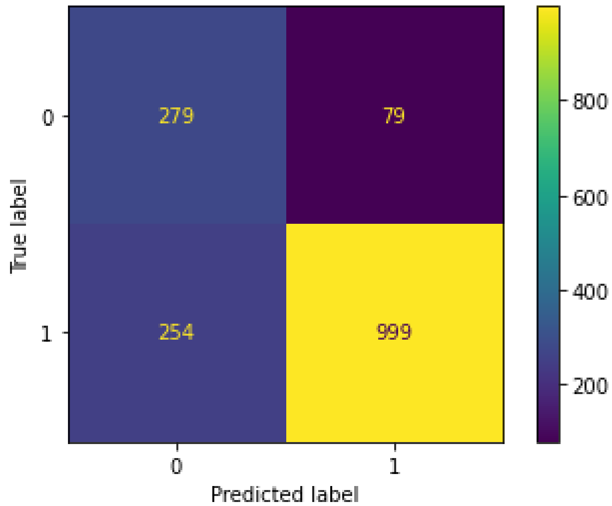
```
Training score:  0.8571428571428571
Validation score:  0.7853658536585366
```

> this was a great starting point, but the model seems to be overfit as the validation score is visible lower than the training score.

creating a confusion matrix from the logistical regression model

In [49]:
```
plot_confusion_matrix(logreg_1, X_train_encoded_df, y_train)
```

Out[49]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7ff4e3054670
>
```




```
In [50]: #there is a visible imbalance within the data set which could be improved by usi
         y_train.value_counts()
```

```
Out[50]: 1    1253
         0     358
         Name: rating_class, dtype: int64
```

Using smote SMOTE on the training set to balance the training dataset

```
In [51]: sm = SMOTE(random_state=42)
```

```
In [52]: #fitting the training set to smote
         X_train_res, y_train_res = sm.fit_resample(X_train_encoded_df, y_train)

         y_train_res.value_counts()
```

```
Out[52]: 1    1253
         0    1253
         Name: rating_class, dtype: int64
```

> the training set is now balanced

Grid Search CV - Model 1: Logistical Regression

```
In [53]: # Looking into class weight.
         # The model will predict positive -
         y.value_counts(normalize=True)
```

```
Out[53]: 1    0.779471
         0    0.220529
         Name: rating_class, dtype: float64
```

```
In [54]: pipe = make_pipeline(SMOTE(random_state=42), LogisticRegression())

         pipe.get_params().keys()
```

```
Out[54]: dict_keys(['memory', 'steps', 'verbose', 'smote', 'logisticregression', 'smote__
         k_neighbors', 'smote__n_jobs', 'smote__random_state', 'smote__sampling_strateg
         y', 'logisticregression__C', 'logisticregression__class_weight', 'logisticregres
```

```
sion__dual', 'logisticregression__fit_intercept', 'logisticregression__intercept
_scaling', 'logisticregression__l1_ratio', 'logisticregression__max_iter', 'logi
sticregression__multi_class', 'logisticregression__n_jobs', 'logisticregression_
_penalty', 'logisticregression__random_state', 'logisticregression__solver', 'lo
gisticregression__tol', 'logisticregression__verbose', 'logisticregression__warm
_start'])
```

In [264…

```
param_grid = {
                'logisticregression__C':[.5,.7,.8,.9,1.0],
                'logisticregression__class_weight':[ { 0:.77, 1:.23}, None, "bal
              'logisticregression__penalty': ['l1', 'l2'],
                'logisticregression__solver': [ 'lbfgs', 'liblinear', 'sag', 'sa
                'logisticregression__max_iter':[100, 1000, 10000]}

gs1 = GridSearchCV(estimator=pipe, param_grid=param_grid, scoring='f1')



gs1.fit(X_train_encoded_df,  y_train)

#returning the best hyperparameters from the search grid
best_params = gs1.best_params_
best_params
```

Out[264…

```
{'logisticregression__C': 1.0,
 'logisticregression__class_weight': None,
 'logisticregression__max_iter': 100,
 'logisticregression__penalty': 'l1',
 'logisticregression__solver': 'liblinear'}
```

In [269…

```
est = gs1.best_estimator_
```

In [270…

```
modeling_function(est, X_train_res, y_train_res, X_val_encoded_df, y_val)
```

```
Training score:  0.9277256022866477
Validation score:  0.8682505399568035
```

Out[270…

```
Pipeline(steps=[('smote', SMOTE(random_state=42)),
                ('logisticregression',
                 LogisticRegression(penalty='l1', solver='liblinear'))])
```

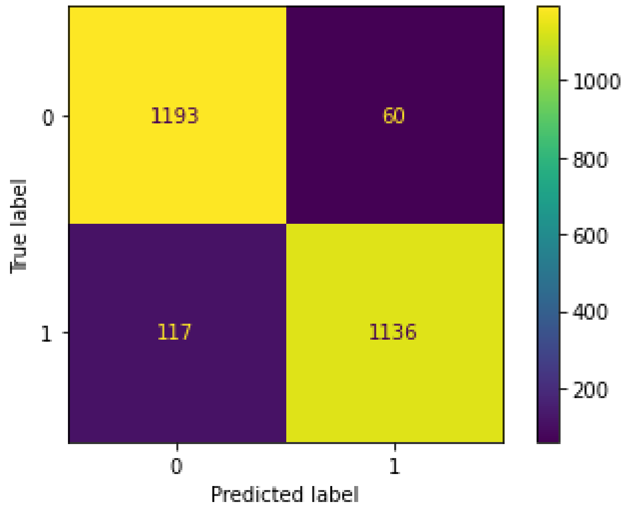
> The model with parameters has a positive outcome; however, the training data is still overfit

In [231…

```
plot_confusion_matrix(est, X_train_res, y_train_res)
```

Out[231…

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f989754e100
>
```



> the model is far more balnced in terms of true negative and true positive after running ther logistical regression model through a pipeline.

Model 2: Decision Tree Classifier

Without Pipeline

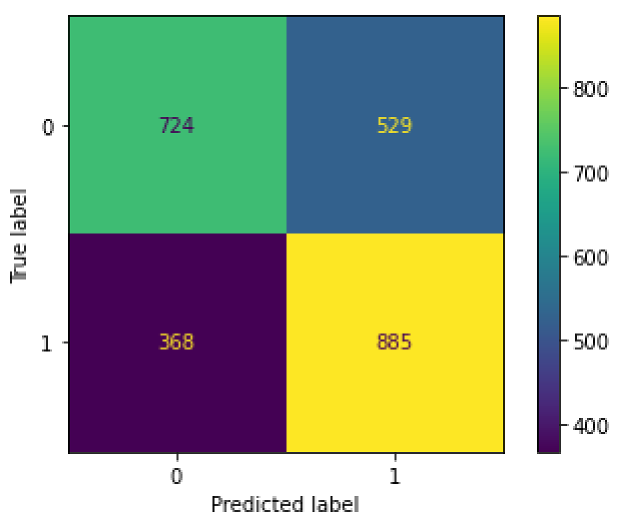
In [254…
```
dt1 = DecisionTreeClassifier(random_state=42, max_depth=2)
```

In [255…
```
dt1 = modeling_function(dt, X_train_res, y_train_res, X_val_encoded_df, y_val)
```

```
Training score:  0.6636670416197976
Validation score:  0.7902439024390244
```

In [406…
```
plot_confusion_matrix(dt, X_train_res, y_train_res)
```

Out[406…
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f987899baf0
>
```



Pipelining Model 2 with Grid Search CV

In [107…
```
pipe2 = make_pipeline(SMOTE(random_state=42), DecisionTreeClassifier())
```

```
pipe2.get_params().keys()
```

Out[107…

```
dict_keys(['memory', 'steps', 'verbose', 'smote', 'decisiontreeclassifier', 'smo
te__k_neighbors', 'smote__n_jobs', 'smote__random_state', 'smote__sampling_strat
egy', 'decisiontreeclassifier__ccp_alpha', 'decisiontreeclassifier__class_weigh
t', 'decisiontreeclassifier__criterion', 'decisiontreeclassifier__max_depth', 'd
ecisiontreeclassifier__max_features', 'decisiontreeclassifier__max_leaf_nodes',
'decisiontreeclassifier__min_impurity_decrease', 'decisiontreeclassifier__min_im
purity_split', 'decisiontreeclassifier__min_samples_leaf', 'decisiontreeclassifi
er__min_samples_split', 'decisiontreeclassifier__min_weight_fraction_leaf', 'dec
isiontreeclassifier__random_state', 'decisiontreeclassifier__splitter'])
```

In [108…

```
param_grid = {'decisiontreeclassifier__criterion' : ['gini', 'entropy'],
              'decisiontreeclassifier__max_depth' : [2,3,4,5,6,7,8,9,10],
              'decisiontreeclassifier__max_leaf_nodes': [2,3,4,5,6,7,8,9,10,11,1

dt1_gs = GridSearchCV(estimator=pipe2, param_grid=param_grid, scoring='f1')
dt1_gs.fit(X_train_encoded_df,  y_train)
best_params = dt1_gs.best_params_
best_params
```

Out[108…

```
{'decisiontreeclassifier__criterion': 'entropy',
 'decisiontreeclassifier__max_depth': 6,
 'decisiontreeclassifier__max_leaf_nodes': 14}
```

In [110…

```
tree_estim = dt1_gs.best_estimator_

mod_2 = modeling_function(tree_estim, X_train_encoded, y_train,
                                X_val_encoded_df, y_val)
```

```
Training score:  0.7759226713532512
Validation score:  0.7468671679197997
```

Model 2 without pipelining, but using Hyperparameters from the Grid Search CV

In [111…

```
best_tree = DecisionTreeClassifier(criterion='entropy', max_depth=5, max_leaf_no
```

In [112…

```
mod_3 = modeling_function(best_tree, X_train_encoded_df, y_train,
                                X_val_encoded_df, y_val)
```

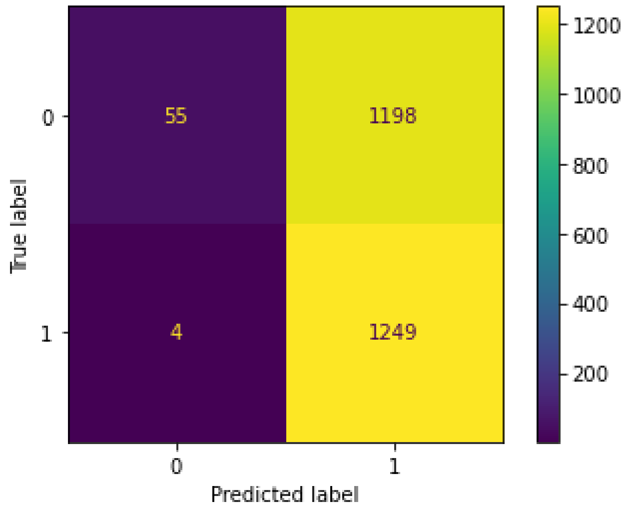
```
Training score:  0.8780316344463973
Validation score:  0.8818897637795275
```

In [113…

```
plot_confusion_matrix(mod_3, X_train_res, y_train_res)
```

Out[113…

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7ff4e24a8a60
>
```



Model 3: Random Forest Classifier

Without Pipeline

In [247…
```
rf1 = RandomForestClassifier(random_state=42)
```

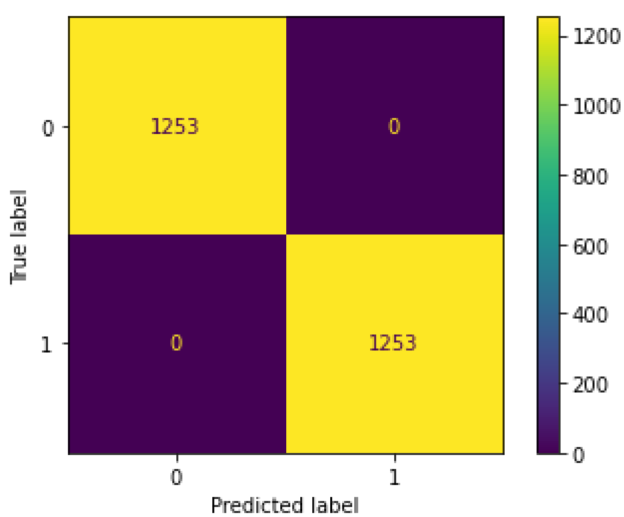
In [248…
```
rf_1 = modeling_function(rf1, X_train_res, y_train_res, X_val_encoded_df, y_val)
```

```
Training score:  1.0
Validation score:  0.8848484848484849
```

> Although it is still overfit, the random forest classifier seems to have an improved outcome in relation to the random tree classifier. It would be interesting to see the outcome with tuned hyperparameters

In [413…
```
plot_confusion_matrix(rf1, X_train_res, y_train_res)
```

Out[413…
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f988688e4f0
>
```



In [233…
```
pipe3 = make_pipeline(SMOTE(random_state=42), RandomForestClassifier())

pipe3.get_params().keys()
```

Out[233…
```
dict_keys(['memory', 'steps', 'verbose', 'smote', 'randomforestclassifier', 'smo
te__k_neighbors', 'smote__n_jobs', 'smote__random_state', 'smote__sampling_strat
egy', 'randomforestclassifier__bootstrap', 'randomforestclassifier__ccp_alpha',
'randomforestclassifier__class_weight', 'randomforestclassifier__criterion', 'ra
ndomforestclassifier__max_depth', 'randomforestclassifier__max_features', 'rando
mforestclassifier__max_leaf_nodes', 'randomforestclassifier__max_samples', 'rand
omforestclassifier__min_impurity_decrease', 'randomforestclassifier__min_impurit
y_split', 'randomforestclassifier__min_samples_leaf', 'randomforestclassifier__m
in_samples_split', 'randomforestclassifier__min_weight_fraction_leaf', 'randomfo
restclassifier__n_estimators', 'randomforestclassifier__n_jobs', 'randomforestcl
assifier__oob_score', 'randomforestclassifier__random_state', 'randomforestclass
ifier__verbose', 'randomforestclassifier__warm_start'])
```

Pipelining Model 3 with Grid Search CV

In [242…
```
#with class weight


param_grid = {'randomforestclassifier__criterion' : ['gini', 'entropy'],
              'randomforestclassifier__max_depth' : [2,3,4,5,6,7,8,9,10],
              'randomforestclassifier__max_leaf_nodes': [2,3,4,5,6,7,8,9,10,11,1
              'randomforestclassifier__class_weight' : [{0:0.78 , 1:0.22}, None,

rf1_gs2 = GridSearchCV(estimator=pipe3, param_grid=param_grid, scoring='f1', n_j
rf1_gs2.fit(X_train_encoded_df,  y_train)
m5_params = rf1_gs2.best_params_
m5_params
```

Out[242…
```
{'randomforestclassifier__class_weight': None,
 'randomforestclassifier__criterion': 'entropy',
 'randomforestclassifier__max_depth': 8,
 'randomforestclassifier__max_leaf_nodes': 13}
```

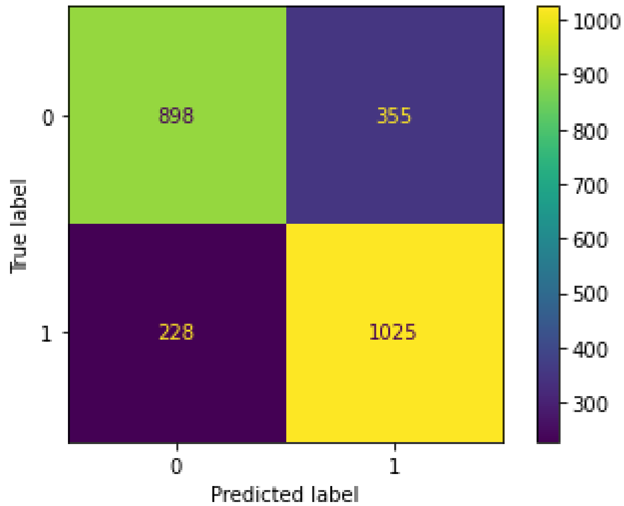
In [416…
```
#pipeline outcome
est3 = rf1_gs2.best_estimator_

modeling_function(est3, X_train_res, y_train_res, X_val_encoded_df, y_val)
```

```
Training score:  0.7886363636363637
Validation score:  0.8194444444444444
```

Out[416…
```
Pipeline(steps=[('smote', SMOTE(random_state=42)),
                ('randomforestclassifier',
                 RandomForestClassifier(criterion='entropy', max_depth=8,
                                        max_leaf_nodes=13))])
```

In [415…
```
plot_confusion_matrix(est3, X_train_res, y_train_res)
```

Out[415…
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f987b8a8700
>
```



Final Model

The projects best model seemed to be the Pipelined Decision Tree Classifier with a Grid Search CV for determining the best hyperparameters

In [55]:
```
X_test_imputed = pd.DataFrame(imputer.transform(X_test), columns=X.columns)
X_test_encoded = ohe.transform(X_test_imputed)
X_test_encoded_df = pd.DataFrame(X_test_encoded, columns=ohe.get_feature_names()
```

In [56]:
```
pipe5 = make_pipeline(SMOTE(random_state=42), DecisionTreeClassifier())

pipe5.get_params().keys()
```

Out[56]:
```
dict_keys(['memory', 'steps', 'verbose', 'smote', 'decisiontreeclassifier', 'smo
te__k_neighbors', 'smote__n_jobs', 'smote__random_state', 'smote__sampling_strat
egy', 'decisiontreeclassifier__ccp_alpha', 'decisiontreeclassifier__class_weigh
t', 'decisiontreeclassifier__criterion', 'decisiontreeclassifier__max_depth', 'd
ecisiontreeclassifier__max_features', 'decisiontreeclassifier__max_leaf_nodes',
'decisiontreeclassifier__min_impurity_decrease', 'decisiontreeclassifier__min_im
purity_split', 'decisiontreeclassifier__min_samples_leaf', 'decisiontreeclassifi
er__min_samples_split', 'decisiontreeclassifier__min_weight_fraction_leaf', 'dec
isiontreeclassifier__random_state', 'decisiontreeclassifier__splitter'])
```

In [98]:
```
param_grid = {'decisiontreeclassifier__criterion' : ['gini', 'entropy'],
              'decisiontreeclassifier__max_depth' : [2,3,4,5,6,7,8,9,10],
              'decisiontreeclassifier__max_leaf_nodes': [2,3,4,5,6,7,8,9,10,11,1

final_dt_gs = GridSearchCV(estimator=pipe5, param_grid=param_grid, scoring='f1')
final_dt_gs.fit(X_train_encoded_df, y_train)
```

Out[98]:
```
GridSearchCV(estimator=Pipeline(steps=[('smote', SMOTE(random_state=42)),
                                       ('decisiontreeclassifier',
                                        DecisionTreeClassifier())]),
             param_grid={'decisiontreeclassifier__criterion': ['gini',
                                                               'entropy'],
                         'decisiontreeclassifier__max_depth': [2, 3, 4, 5, 6, 7,
                                                               8, 9, 10],
                         'decisiontreeclassifier__max_leaf_nodes': [2, 3, 4, 5,
                                                                    6, 7, 8, 9,
                                                                    10, 11, 12,
```

```
                                                                           13, 14]},
                 scoring='f1')
```

Out[98]:
```
GridSearchCV(estimator=Pipeline(steps=[('smote', SMOTE(random_state=42)),
                                       ('decisiontreeclassifier',
                                        DecisionTreeClassifier())]),
             param_grid={'decisiontreeclassifier__criterion': ['gini',
                                                                'entropy'],
                         'decisiontreeclassifier__max_depth': [2, 3, 4, 5, 6, 7,
                                                                8, 9, 10],
                         'decisiontreeclassifier__max_leaf_nodes': [2, 3, 4, 5,
                                                                     6, 7, 8, 9,
                                                                     10, 11, 12,
                                                                     13, 14]},
             scoring='f1')
```

Taking the best parameters to plug the testing data into the final model

In [101…
```
#returning the best parameters from the grid search
best_params = final_dt_gs.best_params_
best_params
```

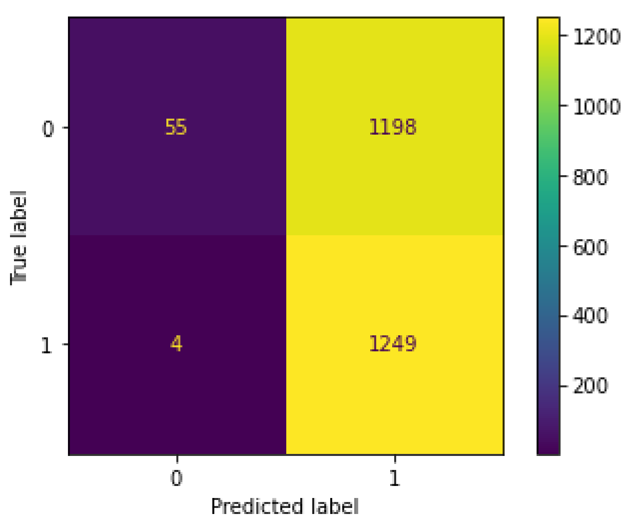
Out[101…
```
{'decisiontreeclassifier__criterion': 'entropy',
 'decisiontreeclassifier__max_depth': 6,
 'decisiontreeclassifier__max_leaf_nodes': 14}
```

In [102…
```
final_mod = DecisionTreeClassifier(criterion='entropy', max_depth=6, max_leaf_no

final_mod.fit(X_train_encoded_df, y_train)
```

Out[102…
```
DecisionTreeClassifier(criterion='entropy', max_depth=6, max_leaf_nodes=14,
                       random_state=42)
```

In [118…
```
plot_confusion_matrix(mod_3, X_train_res, y_train_res)
```

Out[118…
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7ff4e2a0e700
>
```



Final Model Prediction

In [75]:
```
test_preds = final_mod_3.predict(X_test_encoded)

f1_score(y_test, test_preds)
```

Out[75]: 0.8754208754208753

Most Impactful Features

In [99]:

```
X_train.columns
```

Out[99]:

```
Index(['memorable_characteristics_list', 'company_loc_bins',
       'bean_origin_bins', 'weighted_specific_origin_bins', 'ingredient_list',
       'review_date_bin', 'cocoa_bucket', 'comp_manufact_bin'],
      dtype='object')
```

In [130…

```
#Matching the top 5 importances with the column indexes
feature_df = pd.DataFrame(final_mod.feature_importances_, index=X_train_encoded_
feature_df
```

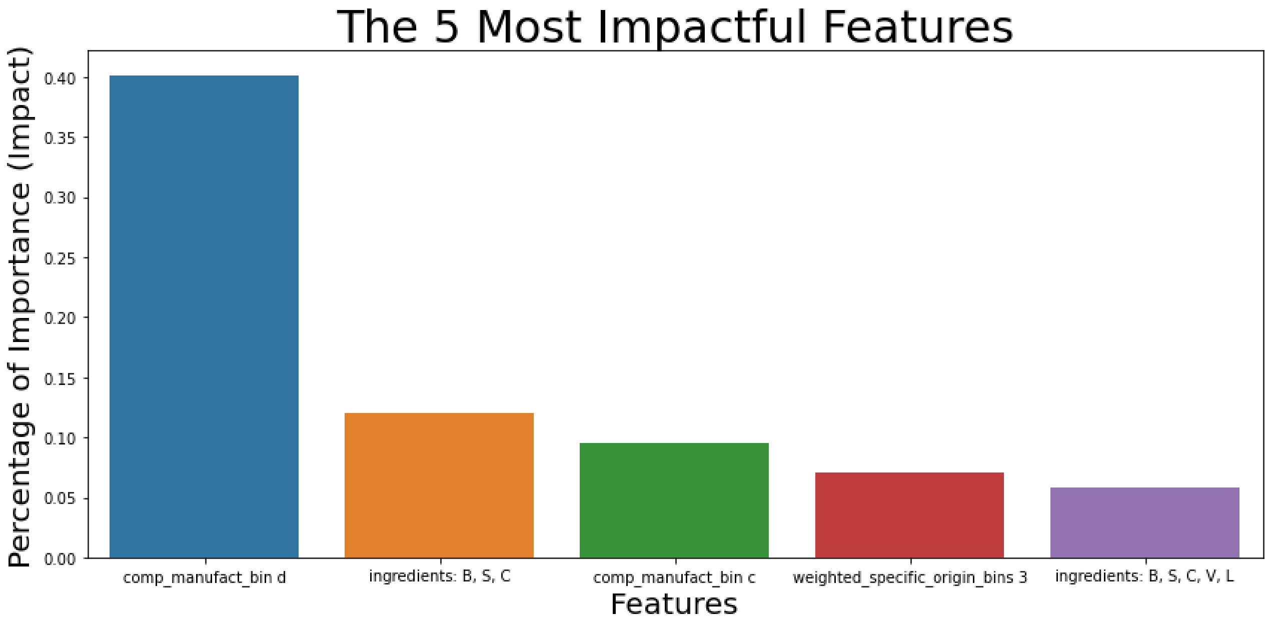
Out[130…

	importance
x7_d	0.397301
x4_B,S,C	0.118924
x7_c	0.094010
x3_3	0.070236
x4_B,S,C,V,L	0.057634
...	...
x0_['floral', 'astringent']	0.000000
x0_['flat', 'metallic', 'floral']	0.000000
x0_['flat', 'late tart notes']	0.000000
x0_['flat', 'floral', 'medicinal']	0.000000
x0_['long lasting', 'cocoa']	0.000000

1619 rows × 1 columns

In [402…

```
fig, ax = plt.subplots(figsize = (12,6))
sns.barplot(data=feature_df.reset_index().iloc[:5], x='index', y='importance', a

ax.set_title('The 5 Most Impactful Features', fontsize=30)
ax.set_xlabel('Features', fontsize=20)
ax.set_ylabel('Percentage of Importance (Impact)', fontsize=20);
ax.set_xticklabels(['comp_manufact_bin d', 'ingredients: B, S, C', 'comp_manufac
plt.tight_layout()
```


The 5 Most Impactful Features
Percentage of Importance (Impact)
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
comp_manufact_bin d
ingredients: B, S, C
comp_manufact_bin c
weighted_specific_origin_bins 3
ingredients: B, S, C, V, L
Features

```
In [ ]:
```