1 Executive Summary

In this notebook, I am going to show why we should use relative weight analysis instead of running a linear regression analysis in some business settings. Also I will discuss the importance of considering the tradeoff between best business outcome and the best machine learning outcome.

Business Problem: We are asked to advise the product design team on the best features of a candy. This recommendation should results in designing a good-selling candy as well.
I am going to use Relative Weight Analysis (both manualy and through "relativeImp" Python Package)n and drive some implications.

2 Introduction

As we know, in a linear regression, importance of features of a product is abstracted from the identified coefficients, using the P-value of each feature to define whether we can take that as reliable or not. However, if predictors are linearly dependent or highly correlated, the OLS becomes unstable.Therefore, we need a tool to tell us how much each feature contributes to criterion variance (R2).
Relative Weight Analysis relies on the decomposition of R2 to assign importance to each predictor. RWA solves this problem by creating predictors that are orthogonal to one another and regressing on these without the effects of multicollinearity. They are then transformed back to the metric of the original predictors.
In its raw form, Relative Weight Analysis returns raw importance scores whose sum equals to the overall R2 of a model; it's normalized form allows us to say “Feature X accounts for Z% of variance in target variable Y.”

3 Importing the Libraries

In [199]:

```
import pandas as pd
import numpy as np
from ipywidgets import interact, interactive, fixed, interact_manual
import ipywidgets as widgets
```

4 Reading the Data

In [200]:

```
df=pd.read_csv('Downloads/candy-data_csv.csv')
df
```

Out[200]:

	competitorname	chocolate	fruity	caramel	peanutyalmondy	nougat	crispedricewafer	hard	bar	pluribus	sugarpercent	pricepercent
0	100 grand	1	0	1	0	0	1	0	1	0	0.732	0.860
1	3 musketeers	1	0	0	0	1	0	0	1	0	0.604	0.511
2	one dime	0	0	0	0	0	0	0	0	0	0.011	0.116
3	one quarter	0	0	0	0	0	0	0	0	0	0.011	0.511
4	air heads	0	1	0	0	0	0	0	0	0	0.906	0.511
...	...	...	...	...	...	...	...	...	...	...	...	...
80	twizzlers	0	1	0	0	0	0	0	0	0	0.220	0.116
81	warheads	0	1	0	0	0	0	1	0	0	0.093	0.116
82	welchs fruit snacks	0	1	0	0	0	0	0	0	1	0.313	0.313
83	werthers original caramel	0	0	1	0	0	0	1	0	0	0.186	0.267
84	whoppers	1	0	0	0	0	1	0	0	1	0.872	0.848

85 rows × 13 columns

In [201]:

```
feature_list= df.columns[1:].to_list()
feature_list
```

Out[201]:

```
['chocolate',
 'fruity',
 'caramel',
 'peanutyalmondy',
 'nougat',
 'crispedricewafer',
 'hard',
 'bar',
 'pluribus',
 'sugarpercent',
 'pricepercent',
 'winpercent']
```

In [202]:
```python
result_df= pd.DataFrame(columns=["Driver","Normalized_RW(Sugarpercent)",
                                 "Normalized_RW(Pricepercent)",
                                 "Normalized_RW(Winpercent)"])
```

In [203]:
```python
result_df.Driver= feature_list[0:-3]
```

5 Relative Weight Analysis

In [204]:
```python
diag[diag_idx] = w_corr_Xs
```

In [205]:
```python
# For each criterion
for i in range(3):

    # Get a correlation between all of the dependent and independent variables.
    corr_matrix = df[feature_list].apply(pd.to_numeric, errors = 'coerce').corr()
    corr_Xs = corr_matrix.iloc[0:-3, 0:-3].copy()
    corr_Xy = corr_matrix.iloc[0:-3, 9 + i].copy()

    # To get around the issue of multi-collinearity
    # Create orthogonal predictors using eigenvectors and eigenvalues on the correlation matrix
    # v_corr_Xs = eigenvector matrix
    w_corr_Xs, v_corr_Xs = np.linalg.eig(corr_Xs)

    # create a diagonal matrix of eigenvalues
    diag_idx = np.diag_indices(len(corr_Xs))

    # Number of features=9
    diag = np.zeros((9, 9), float)
    diag[diag_idx] = w_corr_Xs

    # make the square root of eigenvalues in the diagonal matrix
    delta = np.sqrt(diag)

    #Multiply the eigenvector matrix and its transposition
    coef_xz = v_corr_Xs @ delta @ v_corr_Xs.transpose()

    #To get the partial effect of each independent variable, we apply matrix multiplication
    # to the inverse and correlation matrices
    coef_yz = np.linalg.inv(coef_xz) @ corr_Xy

    #the sum of the squares of coef_yz above is the total sum of the R2!
    r2 = sum(np.square(coef_yz))

    # We calculate the relative weight as the multiplication of the matrix in Step 2 and step 3.
    raw_relative_weights = np.square(coef_xz) @ np.square(coef_yz)

    # The normalized version is then the percentage of r2 that these account for.
    normalized_relative_weights = (raw_relative_weights/r2)*100

    # Adding the result to the daraframe
    result_df.iloc[:,i + 1 ]=normalized_relative_weights.tolist()
```

6 Linking the Results with a Widget

In [206]:
```python
def f(Criterion):
    return (result_df.sort_values(by=Criterion, ascending=True))
```

In [207]:
```
interact(f, Criterion=['Normalized_RW(Sugarpercent)','Normalized_RW(Pricepercent)','Normalized_RW(Winpercent
```

Criterion Normalized_RW(Winpercent)

	Driver	Normalized_RW(Sugarpercent)	Normalized_RW(Pricepercent)	Normalized_RW(Winpercent)
4	nougat	7.416660	3.032290	2.071375
2	caramel	36.358567	6.044282	2.582745
8	pluribus	13.367345	3.659472	2.645129
6	hard	14.294456	4.024305	7.363685
1	fruity	4.222516	11.473136	8.198689
7	bar	6.362093	29.225846	8.436575
5	crispedricewafer	3.201877	10.230328	9.611686
3	peanutyalmondy	6.911266	11.123301	17.207186
0	chocolate	7.865219	21.187039	41.882931

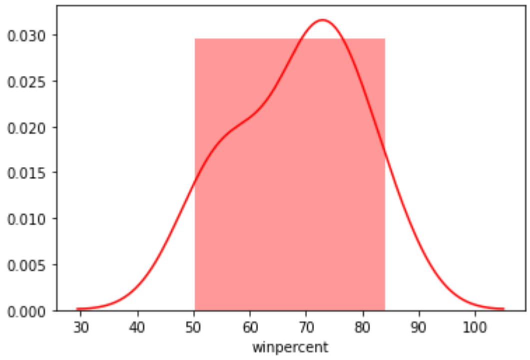
Out[207]: <function __main__.f(Criterion)>

7 Conclusion

Looking at the results, one can easily see that "chocolate" and "peanutyalmondy" flavors wins if we only consider the "Winpercent". This is actually what customer wants but is it the most profitable option for the business? If we change the criterion to "Pricepercent", we can see that the position of "nougat" in the ranking is the lowest.

In [208]:
```
# most wanted features
import seaborn as sns
sns.distplot( df.loc[((df['peanutyalmondy'] == 1) & (df['chocolate'] == 1))]['winpercent'],
             color="red", label="With_Nuts")
```

Out[208]: <matplotlib.axes._subplots.AxesSubplot at 0x1c9fc4a0d88>

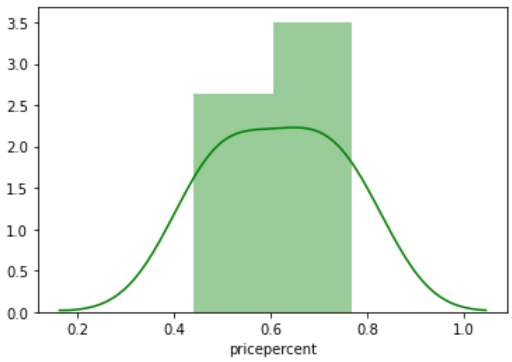


In [209]:
```
# We have 12 of such configurations in our inventory
df.loc[((df['peanutyalmondy'] == 1) & (df['chocolate'] == 1))].count().unique()
```

Out[209]: array([12], dtype=int64)

In [210]:
```
# Leads to the lowest price during the production line
sns.distplot( df.loc[((df['nougat'] == 1) )]['pricepercent'],
             color="green", label="Low_Price")
```

Out[210]: <matplotlib.axes._subplots.AxesSubplot at 0x1c9fd77cb88>

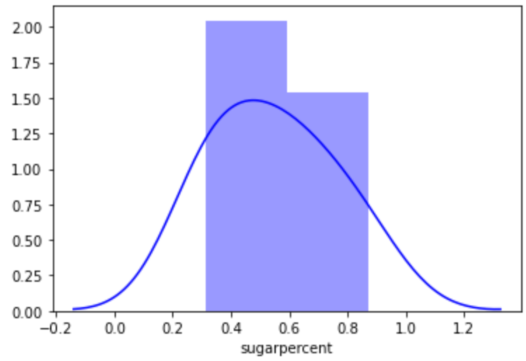


In [211]:
```
# we have 7 of them in our inventory
df.loc[(df['nougat'] == 1)].count().unique()
```

Out[211]: array([7], dtype=int64)

In [212]:
```
# Contains the lowest level of sugar in it
sns.distplot( df.loc[((df['crispedricewafer'] == 1) )]['sugarpercent'],
             color="blue", label="fff")
```

Out[212]: <matplotlib.axes._subplots.AxesSubplot at 0x1c9fd9cea88>



In [213]:
```
# we have 7 of them in our inventory
df.loc[(df['crispedricewafer'] == 1)].count().unique()
```

Out[213]: array([7], dtype=int64)

In [214]:
```
 # we have 0 of them in our inventory
df.loc[((df['nougat'] == 1 ) & (df['crispedricewafer'] == 1) & (df['peanutyalmondy'] == 1)
        & (df['chocolate'] == 1) )].count().unique()
```

Out[214]: array([0], dtype=int64)

This Combination probably has the potential to enter our production line, since it is cheaper, has less sugar and has the highest winning point features.

8 RWA using "relativeImp" Python Package

In [218]:
```
from relativeImp import relativeImp

yName = 'winpercent'
xNames = [
 'chocolate',
 'fruity',
 'caramel',
 'peanutyalmondy',
 'nougat',
 'crispedricewafer',
 'hard',
 'bar',
 'pluribus']

df_results_win = relativeImp(df, outcomeName = yName, driverNames = xNames)
df_results
```

Out[218]:

	driver	rawRelaImpt	normRelaImpt
0	chocolate	0.215617	41.882931
1	fruity	0.042208	8.198689
2	caramel	0.013296	2.582745
3	peanutyalmondy	0.088584	17.207186
4	nougat	0.010664	2.071375
5	crispedricewafer	0.049482	9.611686
6	hard	0.037909	7.363685
7	bar	0.043432	8.436575
8	pluribus	0.013617	2.645129

As we can see, the obtained results are identical, so we can drive the same implications.

Reference: Modified version of the following post

[1] https://towardsdatascience.com/key-driver-analysis-in-python-788beb9b8a7d (https://towardsdatascience.com/key-driver-analysis-in-python-788beb9b8a7d)