Introdution

The dataset I have chosen contains a list of candy. The candy contains features such as chocolate, fruity, peanutyalmondy, nougat, crispedricewafer, hard, bar, pluribus and caramel. These are represented as binary variables, 1 means yes and 0 means no.

The question to be answered in this assignment is:

1) What is the most important feature?

2) Predicting if the candy will be chocolate?

Columns:

1. **chocolate**: Does the candy contain chocolate?
2. **fruity**: Is the candy fruity?
3. **caramel**: Does the candy contain caramel?
4. **peanutalmondy**: Does the candy contain almonds or peanuts?
5. **nougat**: Does it contain nougat
6. **crispedricewafer**: Does it contain crisped rice or wafers?
7. **hard**: Is it a hard candy?
8. **bar**: Is it a candy bar?
9. **pluribus**: Is it one of many candies in a bag or box?
10. **sugarpercent**: The percentile of sugar it falls under within the data set.
11. **pricepercent**: The unit price percentile compared to the rest of the set.
12. **winpercent**: The overall win percentage according to 269,000 matchups.

1) Imports

`Pandas` is a python library. It is used for data manipulation and analysis (Read_csv)

`seaborn` - data visualisation library. (Heatmap)

`matplotlib.pyplot` - embedding plots (Heatmap)

`Sklearn tree` - import decision tree (Tree)

`Sklearn Linear_model` - import linear model (Linear Model)

`sklearn.model_selection import train_test_split` - Creating Train and test set (Train&Test)

`sklearn.linear_model import LogisticRegression` - import logistic regression (LR)

`sklearn import metrics` - Metrics (accuracy)

`klearn.metrics import mean_squared_error` - Test accuracy (Confusion Matrix)

In [17]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn import tree
from sklearn import datasets, linear_model
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
from sklearn.metrics import mean_squared_error, r2_score
```

2) Read in data

- The read_csv function reads in the dataset from github. read_csv can control delimiters, rows, column names. The data is stored in a dataframe `candy`
- drop_duplicates function removes any duplicates

In [18]:
```
candy = pd.read_csv('https://raw.githubusercontent.com/fivethirtyeight/data/master/candy-power-ranking/candy-data.csv')
candy.drop_duplicates(inplace=True)
```

3) Descriptive Statistics

Performing descriptive Statistics help understand the features of the candy data set by giving short summaries and measuress of the data.

What I found:

- `.head()` displays first 5 rows in dataset.
- `.describe()` is used to view some basic statistical details like count, mean, std, min, 25%, 50%, 75% and max
- `.index` returns a list of the indexs
- `.columns` return all columns. There is a total of 13 columns
- `.dtypes` returns the data types of each column. Is used to check if data is categorical or numerical. All data is numerical except for competitorname. The numerical columns are divided into 9 integars and 3 floats
- `.shape` returns a tuple representing the dimensionality of the DataFrame.The result is (85,13) - 85 rows and 13 columns
- `.values` returns the values in the dataframe
- `.info()` prints information including the index dtype and columns, non-null values

In [19]:

```
candy.head()
```

Out[19]:

	competitorname	chocolate	fruity	caramel	peanutyalmondy	nougat	crispedricewafer	hard
0	100 Grand	1	0	1	0	0	1	0
1	3 Musketeers	1	0	0	0	1	0	0
2	One dime	0	0	0	0	0	0	0
3	One quarter	0	0	0	0	0	0	0
4	Air Heads	0	1	0	0	0	0	0

In [20]:
```
candy.describe()
```

Out[20]:

	chocolate	fruity	caramel	peanutyalmondy	nougat	crispedricewafer	hard
count	85.000000	85.000000	85.000000	85.000000	85.000000	85.000000	85.000000
mean	0.435294	0.447059	0.164706	0.164706	0.082353	0.082353	0.176471
std	0.498738	0.500140	0.373116	0.373116	0.276533	0.276533	0.383482
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
50%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
75%	1.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

In [21]:
```
candy.index
```

Out[21]:
```
Int64Index([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14,
15, 16,
            17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31,
32, 33,
            34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48,
49, 50,
            51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65,
66, 67,
            68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82,
83,
            84],
           dtype='int64')
```

In [6]:
```
candy.columns
```

Out[6]:
```
Index(['competitorname', 'chocolate', 'fruity', 'caramel', 'peanutyalmo
ndy',
       'nougat', 'crispedricewafer', 'hard', 'bar', 'pluribus', 'sugarp
ercent',
       'pricepercent', 'winpercent'],
      dtype='object')
```

In [22]:
```
candy.dtypes
```

Out[22]:
```
competitorname       object
chocolate             int64
fruity                int64
caramel               int64
peanutyalmondy        int64
nougat                int64
crispedricewafer      int64
hard                  int64
bar                   int64
pluribus              int64
sugarpercent        float64
pricepercent        float64
winpercent          float64
dtype: object
```

In [23]:
```
candy.shape
```

Out[23]:
```
(85, 13)
```

In [24]:
```
candy.values
```

Out[24]:
```
array([['100 Grand', 1, 0, ..., 0.73199999, 0.86000001, 66.971725],
       ['3 Musketeers', 1, 0, ..., 0.60399997, 0.51099998, 67.602936],
       ['One dime', 0, 0, ..., 0.011000000000000001, 0.11599999999999999
9,
        32.261086],
       ...,
       ['WelchÕs Fruit Snacks', 0, 1, ..., 0.31299999, 0.31299999,
        44.375519],
       ['WertherÕs Original Caramel', 0, 0, ..., 0.18600000000000003,
        0.26699999, 41.904308],
       ['Whoppers', 1, 0, ..., 0.87199998, 0.84799999, 49.524113]],
      dtype=object)
```

In [25]:
```
candy.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 85 entries, 0 to 84
Data columns (total 13 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   competitorname    85 non-null     object 
 1   chocolate         85 non-null     int64  
 2   fruity            85 non-null     int64  
 3   caramel           85 non-null     int64  
 4   peanutyalmondy    85 non-null     int64  
 5   nougat            85 non-null     int64  
 6   crispedricewafer  85 non-null     int64  
 7   hard              85 non-null     int64  
 8   bar               85 non-null     int64  
 9   pluribus          85 non-null     int64  
 10  sugarpercent      85 non-null     float64
 11  pricepercent      85 non-null     float64
 12  winpercent        85 non-null     float64
dtypes: float64(3), int64(9), object(1)
memory usage: 9.3+ KB
```

3) More Descriptive Statistics

- `.sugarpercent.describe()` views the view some basic statistical details like count, mean, std, min, 25%, 50%, 75% and max on sugarpercent
- `.sugarpercent.mean()` will find the average percentage of sugar in each candy
- `.sugarpercent.max()` - max value of sugar percentage
- `.sugarpercent.min()` - min value of sugar percentage
- `candy[candy.sugarpercent > candy.sugarpercent.mean()].sort_values(by=['sugarpercent'],ascending=False)` - shows candies that are higher then the mean. sorts by top sugar percent. Top 5 are ReeseÕs stuffed with pieces, Sugar Babies, Milky Way Simply Caramel and Skittles wildberry
- `candy[candy.sugarpercent <= candy.sugarpercent.mean()].sort_values(by=['sugarpercent'],ascending=True)` - shows candies that are lower then the mean. sorts by lowest sugar percent. The candies with the lowest sugar percentage are one dime, one quarter, ReeseÕs Miniatures, Chiclets and lemonhead
- `candy[candy.sugarpercent==candy.sugarpercent.max()].competitorname` - Candy with the highest sugar content = ReeseÕs stuffed with pieces
- `candy[candy['fruity']==0].sort_values(by=['winpercent'], ascending=False).head(10)` - sorts non fruity candies by winpercent. Top non fruity candies are ReeseÕs Peanut Butter cup, ReeseÕs Miniatures, Twix, Kit Kat and Snickers
- `candy[(candy['caramel']==1)&(candy['fruity']==1)]` - displays candy which is both fruity and has caramel = Caramel Apple Pops
- `sort = candy[['competitorname','winpercent']].sort_values(by='winpercent') pd.concat([sort.head(5),sort.tail(5)],axis=0).plot(x='competitorname',y='winperce popularity',sort_columns=True,figsize = (5,5))` - Data visualisation showing the most popular candies(Bar chart)

In [31]:
```
candy.sugarpercent.describe()
```

Out[31]:
```
count    85.000000
mean      0.478647
std       0.282778
min       0.011000
25%       0.220000
50%       0.465000
75%       0.732000
max       0.988000
Name: sugarpercent, dtype: float64
```

In [32]:
```
candy.sugarpercent.mean()
```

Out[32]:
```
0.4786470514588237
```

In [33]:
```
candy.sugarpercent.max()
```
Out[33]:
```
0.98799998
```

In [34]:
```
candy.sugarpercent.min()
```
Out[34]:
```
0.011000000000000001
```

In [35]:
```
candy[candy.sugarpercent > candy.sugarpercent.mean()].sort_values(by=['s
ugarpercent'],ascending=False)
```

Out[35]:

	competitorname	chocolate	fruity	caramel	peanutyalmondy	nougat	crispedricewafer	hard
54	ReeseÕs stuffed with pieces	1	0	0	1	0	0	0
70	Sugar Babies	0	0	1	0	0	0	0
38	Milky Way Simply Caramel	1	0	1	0	0	0	0
61	Skittles wildberry	0	1	0	0	0	0	0
60	Skittles original	0	1	0	0	0	0	0
4	Air Heads	0	1	0	0	0	0	0
8	Candy Corn	0	0	0	0	0	0	0
17	Gobstopper	0	1	0	0	0	0	1
84	Whoppers	1	0	0	0	0	1	0
34	Mike & Ike	0	1	0	0	0	0	0
58	Runts	0	1	0	0	0	0	1
56	Rolo	1	0	1	0	0	0	0
41	Nerds	0	1	0	0	0	0	1
33	M&MÕs	1	0	0	0	0	0	0
32	Peanut butter M&MÕs	1	0	0	1	0	0	0
55	Ring pop	0	1	0	0	0	0	1
57	Root Beer Barrels	0	0	0	0	0	0	1
0	100 Grand	1	0	1	0	0	1	0
37	Milky Way Midnight	1	0	1	0	1	0	0
16	Fun Dip	0	1	0	0	0	0	1
13	Dots	0	1	0	0	0	0	0
14	Dum Dums	0	1	0	0	0	0	1
11	Chewey Lemonhead Fruit Mix	0	1	0	0	0	0	0
52	ReeseÕs Peanut Butter cup	1	0	0	1	0	0	0
36	Milky Way	1	0	1	0	1	0	0
49	Pop Rocks	0	1	0	0	0	0	1
1	3 Musketeers	1	0	0	0	1	0	0
10	Charleston Chew	1	0	0	0	1	0	0
9	Caramel Apple Pops	0	1	1	0	0	0	0
65	Snickers Crisper	1	0	1	1	0	1	0

	competitorname	chocolate	fruity	caramel	peanutyalmondy	nougat	crispedricewafer	hard
6	Baby Ruth	1	0	1	1	1	0	0
73	Swedish Fish	0	1	0	0	0	0	0
74	Tootsie Pop	1	1	0	0	0	0	1
42	Nestle Butterfinger	1	0	0	1	0	0	0
47	Peanut M&Ms	1	0	0	1	0	0	0
50	Red vines	0	1	0	0	0	0	0
69	Strawberry bon bons	0	1	0	0	0	0	1
64	Snickers	1	0	1	1	1	0	0
79	Twix	1	0	1	0	0	1	0

In [36]:
```
candy[candy.sugarpercent <= candy.sugarpercent.mean()].sort_values(by=[
'sugarpercent'],ascending=True)
```

Out[36]:

	competitorname	chocolate	fruity	caramel	peanutyalmondy	nougat	crispedricewafer	hard
2	One dime	0	0	0	0	0	0	0
3	One quarter	0	0	0	0	0	0	0
51	ReeseÕs Miniatures	1	0	0	1	0	0	0
12	Chiclets	0	1	0	0	0	0	0
30	Lemonhead	0	1	0	0	0	0	1
67	Sour Patch Tricksters	0	1	0	0	0	0	0
66	Sour Patch Kids	0	1	0	0	0	0	0
81	Warheads	0	1	0	0	0	0	1
48	Pixie Sticks	0	0	0	0	0	0	0
26	Jawbusters	0	1	0	0	0	0	1
15	Fruit Chews	0	1	0	0	0	0	0
22	HersheyÕs Kisses	1	0	0	0	0	0	0
68	Starburst	0	1	0	0	0	0	0
72	Super Bubble	0	1	0	0	0	0	0
76	Tootsie Roll Midgies	1	0	0	0	0	0	0
83	WertherÕs Original Caramel	0	0	1	0	0	0	1
44	Nik L Nip	0	1	0	0	0	0	0
27	Junior Mints	1	0	0	0	0	0	0
59	Sixlets	1	0	0	0	0	0	0
29	Laffy Taffy	0	1	0	0	0	0	0
80	Twizzlers	0	1	0	0	0	0	0
45	Now & Later	0	1	0	0	0	0	0
31	Lifesavers big ring gummies	0	1	0	0	0	0	0
63	Smarties candy	0	1	0	0	0	0	1
62	Nestle Smarties	1	0	0	0	0	0	0
35	Milk Duds	1	0	1	0	0	0	0
78	Trolli Sour Bites	0	1	0	0	0	0	0
7	Boston Baked Beans	0	0	0	1	0	0	0
75	Tootsie Roll Juniors	1	0	0	0	0	0	0
28	Kit Kat	1	0	0	0	0	1	0

	competitorname	chocolate	fruity	caramel	peanutyalmondy	nougat	crispedricewafer	hard
43	Nestle Crunch	1	0	0	0	0	1	0
82	WelchÕs Fruit Snacks	0	1	0	0	0	0	0
39	Mounds	1	0	0	0	0	0	0
40	Mr Good Bar	1	0	0	1	0	0	0
53	ReeseÕs pieces	1	0	0	1	0	0	0
71	Sugar Daddy	0	0	1	0	0	0	0
23	HersheyÕs Krackel	1	0	0	0	0	1	0
24	HersheyÕs Milk Chocolate	1	0	0	0	0	0	0
25	HersheyÕs Special Dark	1	0	0	0	0	0	0
46	Payday	0	0	0	1	1	0	0
21	Haribo Twin Snakes	0	1	0	0	0	0	0
20	Haribo Sour Bears	0	1	0	0	0	0	0
19	Haribo Happy Cola	0	0	0	0	0	0	0
18	Haribo Gold Bears	0	1	0	0	0	0	0
77	Tootsie Roll Snack Bars	1	0	0	0	0	0	0
5	Almond Joy	1	0	0	1	0	0	0

In [37]:
```
candy[candy.sugarpercent==candy.sugarpercent.max()].competitorname
```

Out[37]:
```
54    ReeseÕs stuffed with pieces
Name: competitorname, dtype: object
```

In [38]:
```
candy[candy['fruity']==0].sort_values(by=['winpercent'], ascending=False
).head(10)
```

Out[38]:

	competitorname	chocolate	fruity	caramel	peanutyalmondy	nougat	crispedricewafer	hard
52	ReeseÕs Peanut Butter cup	1	0	0	1	0	0	0
51	ReeseÕs Miniatures	1	0	0	1	0	0	0
79	Twix	1	0	1	0	0	1	0
28	Kit Kat	1	0	0	0	0	1	0
64	Snickers	1	0	1	1	1	0	0
53	ReeseÕs pieces	1	0	0	1	0	0	0
36	Milky Way	1	0	1	0	1	0	0
54	ReeseÕs stuffed with pieces	1	0	0	1	0	0	0
32	Peanut butter M&MÕs	1	0	0	1	0	0	0
42	Nestle Butterfinger	1	0	0	1	0	0	0

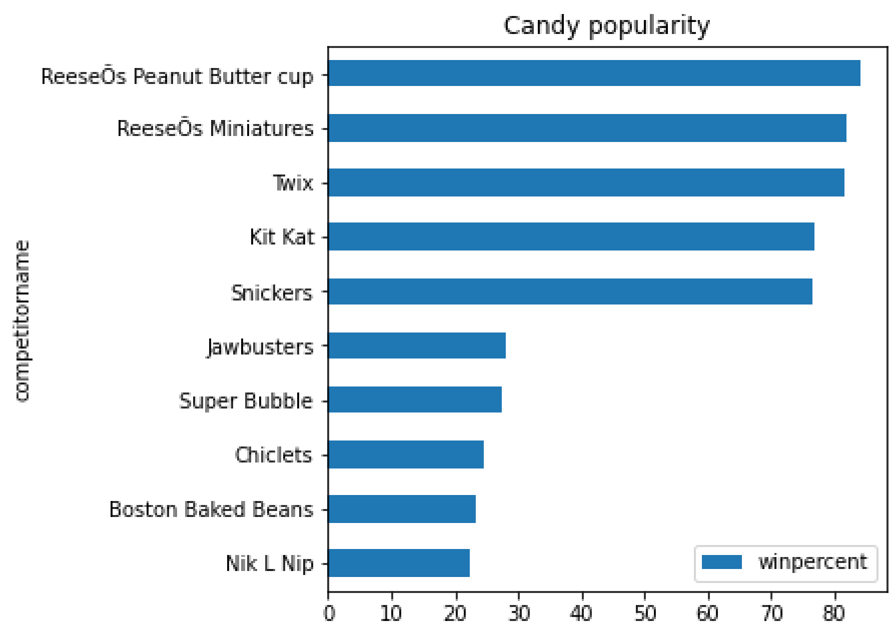
In [39]:
```
candy[(candy['caramel']==1)&(candy['fruity']==1)]
```

Out[39]:

	competitorname	chocolate	fruity	caramel	peanutyalmondy	nougat	crispedricewafer	hard
9	Caramel Apple Pops	0	1	1	0	0	0	0

In [40]:
```
sort = candy[['competitorname','winpercent']].sort_values(by='winpercen
t')
pd.concat([sort.head(5),sort.tail(5)],axis=0).plot(x='competitorname',y=
'winpercent',kind='barh',title='Candy popularity',sort_columns=True,figs
ize = (5,5))
```

Out[40]: <matplotlib.axes._subplots.AxesSubplot at 0x7f7582e45f60>



4) Exploratory Data Analysis

a) Correlation

Multivariative analysis

This is used to find the pairwise correlation of all the columns in the dataframe i.e. finds relationship between the features.

By looking at the correlation we can easily identify useful exporatory variables.

A negative pair number on the chart means the two candies are less likely both have those attributes

A positive pair number on the chart means the two candies are more likely both have those attributes

High Correlations between features normally results in poor linear and logistic Regression performance

In this plot we can see that chocolate candies are rarely fruity

Correlation lets us check for multicollinearity. The coefficients in this plot are normal and stable so we do not have to worry about this

A heat map is a visualisation technique where each value is depicted by colour.Bisically it will show that get the most attention.

In my heatmap it ranges from dark blue to dark red. The more negative the value darker the shade of blue. The more positive the darker the shade of red.

This heatmap is used to make it even easier to identify the correlations

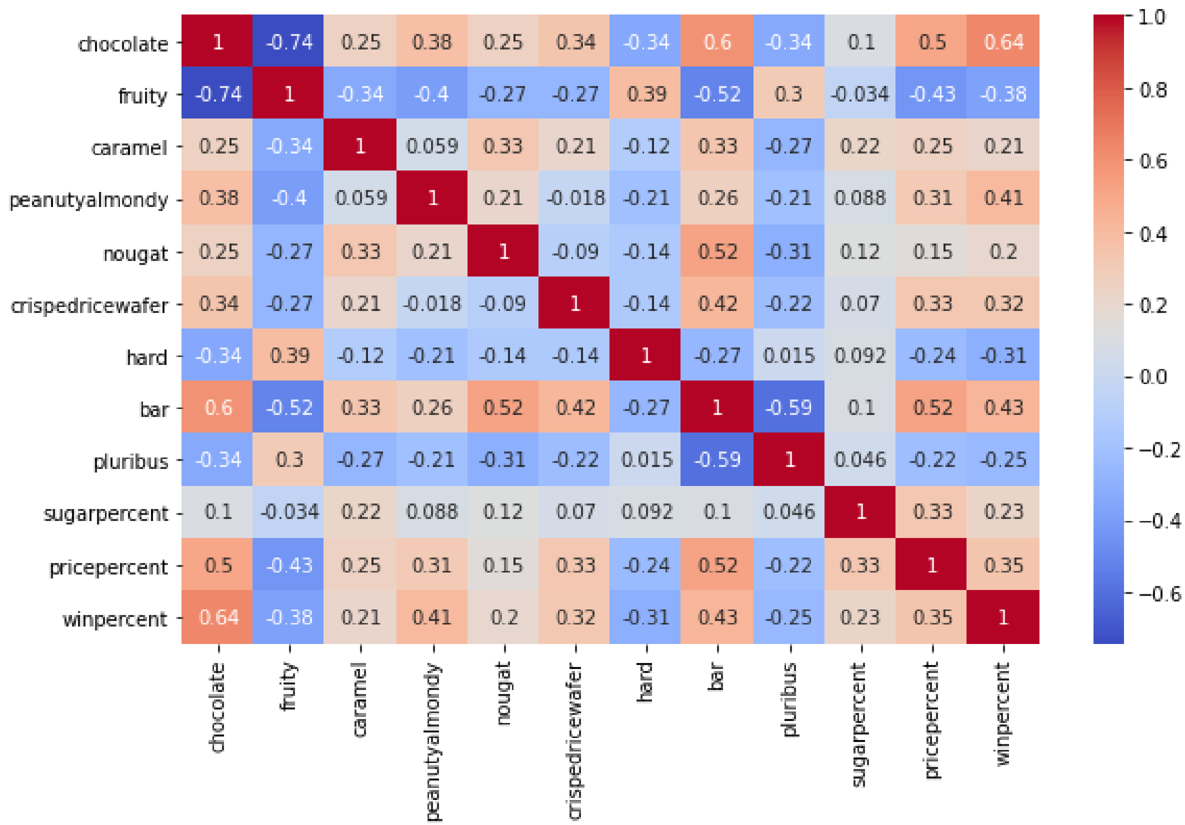
In [41]:
```
Correlations=candy.corr()
Correlations
```

Out[41]:

	chocolate	fruity	caramel	peanutyalmondy	nougat	crispedricewafer
chocolate	1.000000	-0.741721	0.249875	0.377824	0.254892	0.341210
fruity	-0.741721	1.000000	-0.335485	-0.399280	-0.269367	-0.269367
caramel	0.249875	-0.335485	1.000000	0.059356	0.328493	0.213113
peanutyalmondy	0.377824	-0.399280	0.059356	1.000000	0.213113	-0.017646
nougat	0.254892	-0.269367	0.328493	0.213113	1.000000	-0.089744
crispedricewafer	0.341210	-0.269367	0.213113	-0.017646	-0.089744	1.000000
hard	-0.344177	0.390678	-0.122355	-0.205557	-0.138675	-0.138675
bar	0.597421	-0.515066	0.333960	0.260420	0.522976	0.423751
pluribus	-0.339675	0.299725	-0.269585	-0.206109	-0.310339	-0.224693
sugarpercent	0.104169	-0.034393	0.221933	0.087889	0.123081	0.069950
pricepercent	0.504675	-0.430969	0.254327	0.309153	0.153196	0.328265
winpercent	0.636517	-0.380938	0.213416	0.406192	0.199375	0.324680

In [42]:
```
plt.figure(figsize = (10,6))
sns.heatmap(candy.corr(),annot=True, cmap = 'coolwarm')
```

Out[42]: <matplotlib.axes._subplots.AxesSubplot at 0x7f7582e457f0>



B) Density Plots

Also known as Kernel Density Plots or Density Trace Graph. A Density Plot visualises the distribution of data over a continuous interval or time period. The peaks of these densities can show where the values are conncentrated.

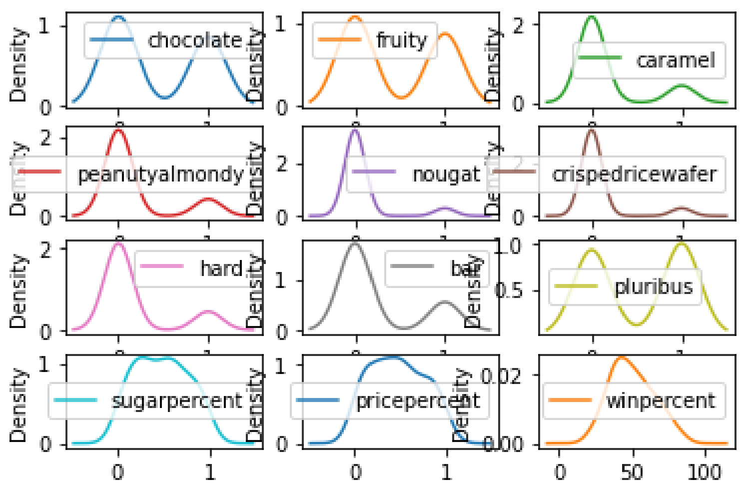
An advantage of density plots over histograms is that it is better at determining the distribution shape. This is beacuse it is not affected by the bins.

By plotting a density plot we change the kind to 'density'

From this plot we can see:

- More candies have no caramel
- more candies are chocolatey
- Nearly all candies are not hard

In [43]:
```
K=candy.plot(kind='density',subplots=True,layout=(4,3),sharex=False,shar
ey=False)
```



C) Box plots

Univariate Analysis

Boxplots summarize the distribution of each attribute, drawing a line for the median. They graphically depict groups of numerical data through their quartiles

Gives an idea of data spread

The winpercent.quantile is set to 60% and above. So if the relationship is above this value. The feature can be considered important.

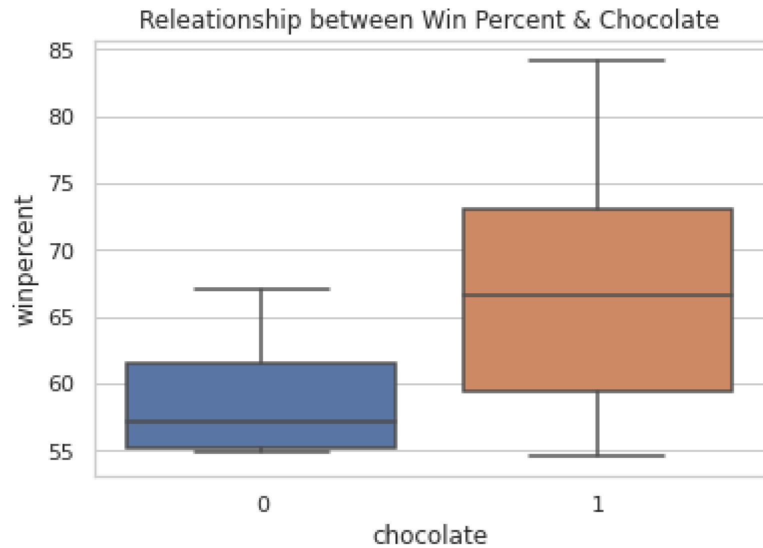
Results of boxplots:

- Win Percent & Chocolate = `<60%` like chocolate candies, Therefore liking choocolate might be an important parameter
- Win Percent & Fruity = `<60%` like fruity candies, Therefore liking fruity candies might be an important parameter
- Win Percent & Caramel = `<60%` like caramel candies, Therefore liking carmel candies might be an important parameter
- Win Percent & PeanutAlmond = `<60%` like peanutalmond candies, Therefore liking peanutalmond candies might be an important parameter
- Win Percent & Nougat = `<60%` like Nougat candies, Therefore liking peanutalmond candies might be an important parameter
- Win Percent & Wafer = `<60%` like wafer candies, Therefore liking Wafer candies might be an important parameter
- Win Percent & Hardness = `<60%` like soft candies, Therefore Soft Candies might be important
- Win Percent & Bar = `<60%` are bars, Therefore being a bar might be an important parameter
- Win Percent & Pluribus = `<60%` , Not clear result

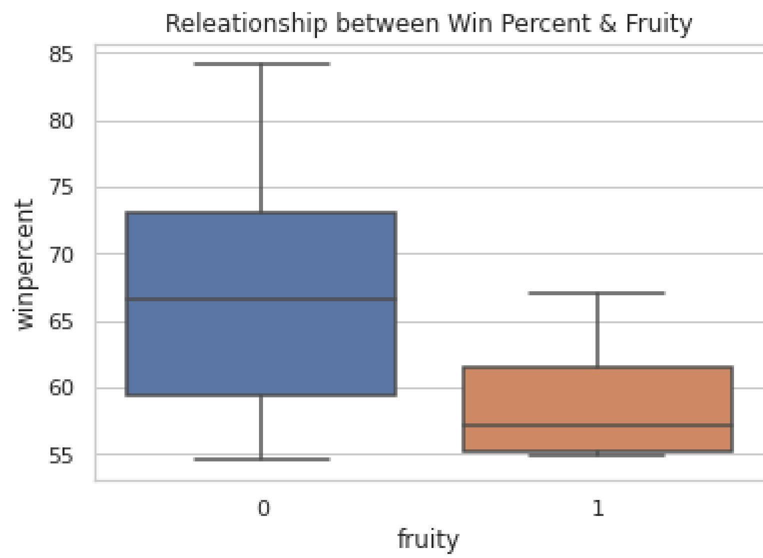
In [44]:
```
win_num = candy[candy.winpercent>candy.winpercent.quantile(.6)]
```

In [45]:
```
sns.set_theme(style="whitegrid")
sns.boxplot(x="chocolate", y="winpercent", data=win_num).set_title('Rele
ationship between Win Percent & Chocolate')
```

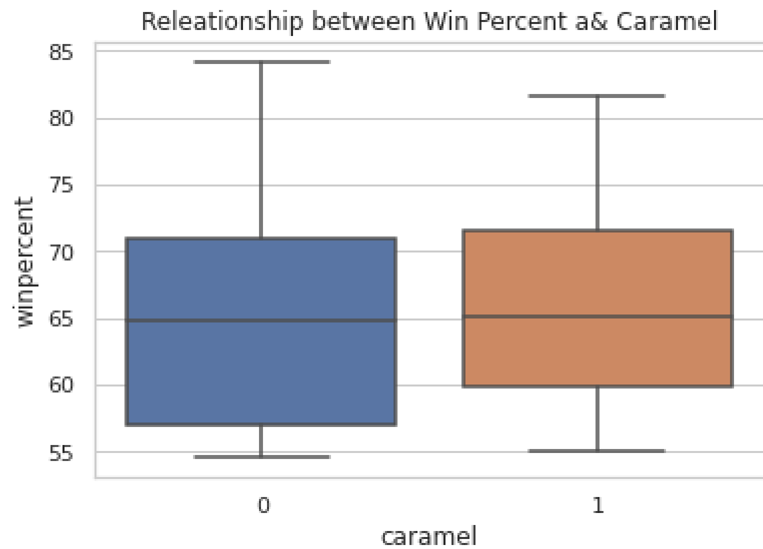
Out[45]: Text(0.5, 1.0, 'Releationship between Win Percent & Chocolate')



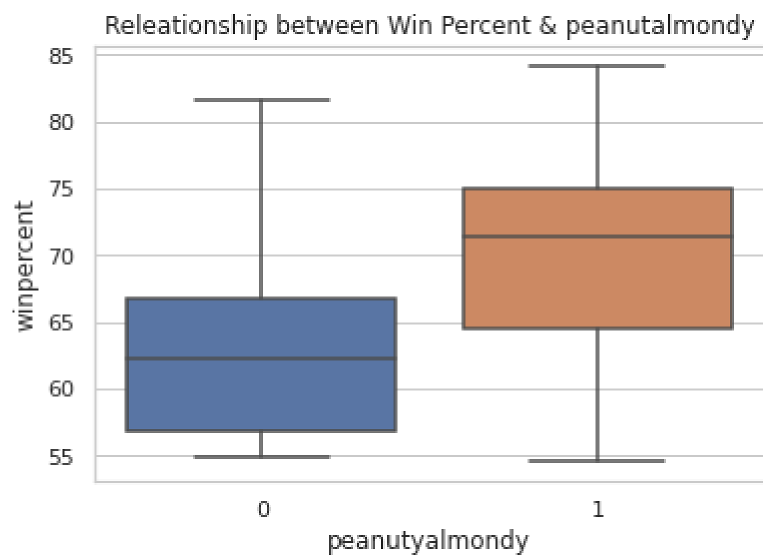
In [46]:
```
sns.boxplot(x="fruity", y="winpercent", data=win_num).set_title('Releati
onship between Win Percent & Fruity');
```



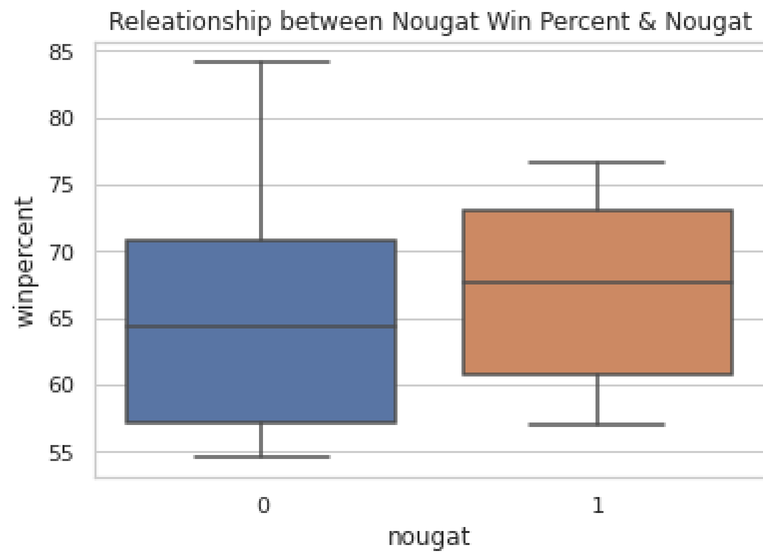
In [47]:
```
sns.boxplot(x="caramel", y="winpercent", data=win_num).set_title('Releat
ionship between Win Percent a& Caramel');
```



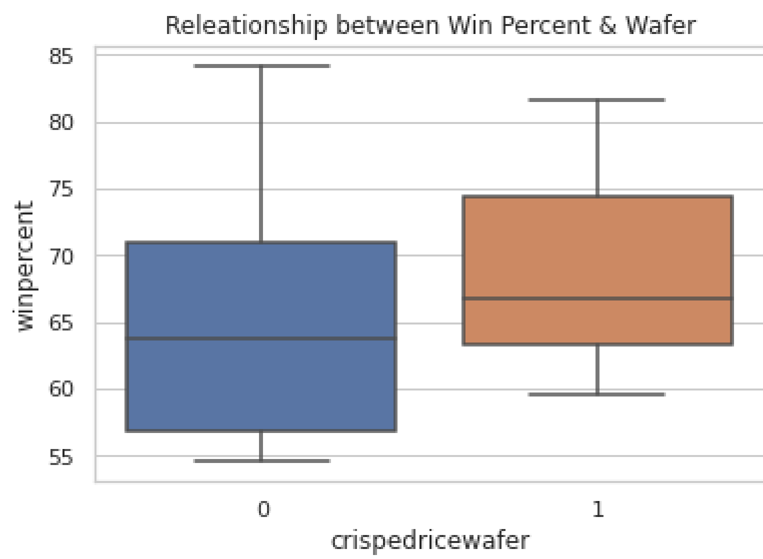
In [48]:
```
sns.boxplot(x="peanutyalmondy", y="winpercent", data=win_num).set_title(
'Releationship between Win Percent & peanutalmondy');
```



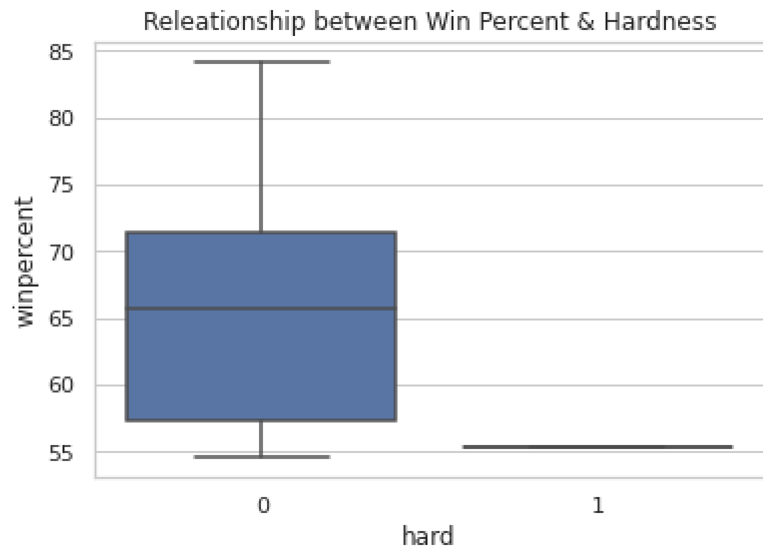
In [49]:
```
sns.boxplot(x="nougat", y="winpercent", data=win_num).set_title('Releati
onship between Nougat Win Percent & Nougat');
```



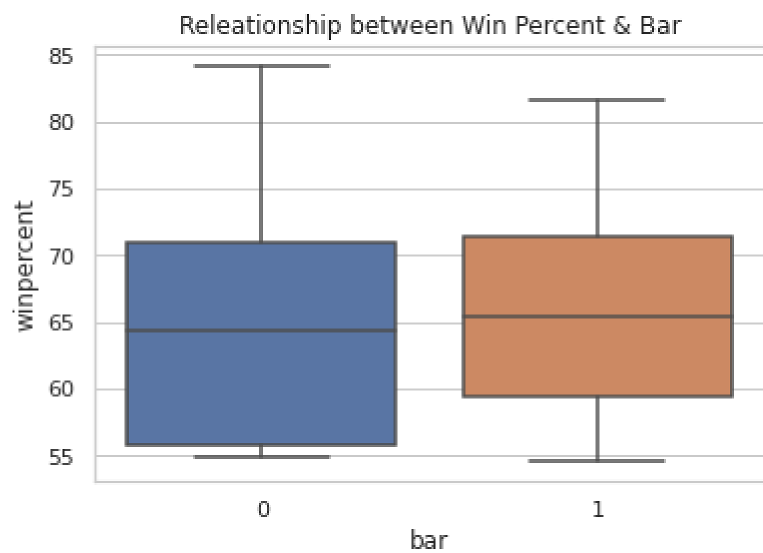
In [50]:
```
sns.boxplot(x="crispedricewafer", y="winpercent", data=win_num).set_titl
e('Releationship between Win Percent & Wafer');
```



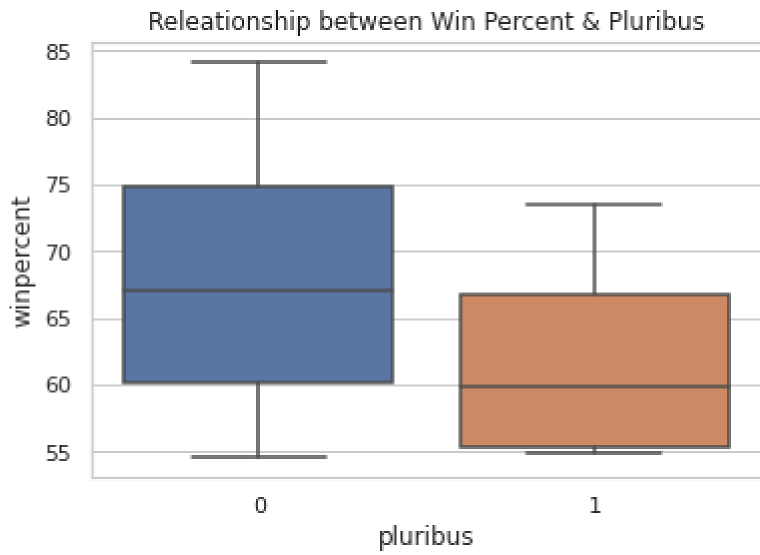
In [51]:
```
sns.boxplot(x="hard", y="winpercent", data=win_num).set_title('Releation
ship between Win Percent & Hardness');
```



In [52]:
```
sns.boxplot(x="bar", y="winpercent", data=win_num).set_title('Releations
hip between Win Percent & Bar');
```



In [53]:
```
sns.boxplot(x="pluribus", y="winpercent", data=win_num).set_title('Relea
tionship between Win Percent & Pluribus');
```



5) Machine Learning

a) Decision Tree

The decision tree alogrithm is used here to visually represent feature importance.

The tree has considered all features but is only display with an importance greater then 0.

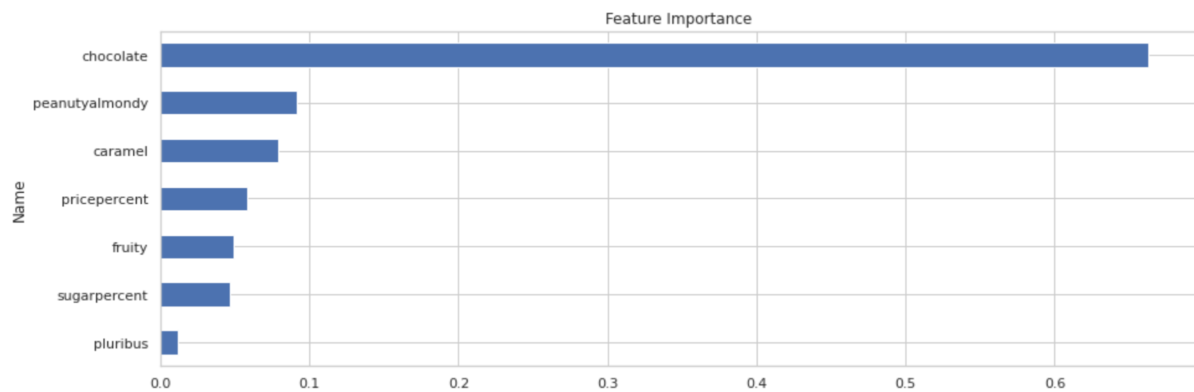
The decision tree is plotted on a bar chart.

From this bar chart we can see that the most important feature is chocolate and the least is pluribus. Chocolate is in fact significantly more important then the next most important peanutalmondy

Chocolate = 0.772891 vs peanutalmondy = 0.091855

In [54]:
```
d_tree = tree.DecisionTreeRegressor(max_depth=3).fit(candy[candy.columns
[1:-1]],candy[candy.columns[-1]])
df_imp = pd.DataFrame.from_dict({'Name':candy.columns[1:-1],'Importance'
:d_tree.feature_importances_})
df_imp_plt = df_imp.sort_values(by='Importance',ascending=True).reset_in
dex(drop=True)
df_imp_plt[df_imp_plt.Importance>0].plot(kind='barh',x='Name',y='Importa
nce',title='Feature Importance',sort_columns=True,figsize = (15,5),legen
d=False)
```

Out[54]: <matplotlib.axes._subplots.AxesSubplot at 0x7f75815732e8>



In [55]:
```
df_imp
```

Out[55]:

	Name	Importance
0	chocolate	0.662891
1	fruity	0.049499
2	caramel	0.079350
3	peanutyalmondy	0.091855
4	nougat	0.000000
5	crispedricewafer	0.000000
6	hard	0.000000
7	bar	0.000000
8	pluribus	0.012091
9	sugarpercent	0.046238
10	pricepercent	0.058076

b) Logistic regression

Logistic Regression is the modelling approach I am taking. It is the bestfitting and least restrictive model.

It describes the relationship between each independent explanatory variables and the dependent binomial response variable.

It will be trying to predict if a candy is chocolatey or not based on all the other features in the dataset

The reason logistic regression is chosen is because it is especially good at predicting boolean values like True or False and has good accuracy for simple datasets.

1. Create a dataframe without the competitor name column.

The competitor name column is a categorical column which will not be used in the machine learning algorithm. It has a unique value in every row so it is impossible to fit a linear model with it. The dataframe is assigned to `X`

2. Create a series with the Chocolate column.

`y` is the respones variable and contains the Choclate values. Also know as the explanatory variable

In [56]:
```
X = candy.drop(['competitorname'], axis = 1)
y = candy['chocolate']
```

3. Create a test and train set.

`train_test_split` splits the candy set. 75% is allocated to the train set and 25% is allocated to the test set. The train set is a sample of data used to fit the model. The test set is used to provide an unbiased evaluation of the final model fit on the training dataset.

In [57]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25
, random_state=0)
```

4. Perform Logistic Regression.

Assign `logisticRegression()` to logreg then fit the model with `logreg.fit()`. The logitic regression is performed on the train set.

In [58]:
```
logreg = LogisticRegression()

logreg.fit(X_train,y_train)
```

Out[58]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=
True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbo
se=0,
                   warm_start=False)
```

5. Prediction.

`logreg.predict()` returns a array of the predicted values. The value 1 means the candy is chocolate. The value O means the candy is not chocolate. The test set has 22 rows. 8 of the values equal 1 and 14 equal 0. We can test this against the original test set and see which values are different. It appears that the 3rd last value is different in each series.

In [59]:
```
y_pred=logreg.predict(X_test)
y_pred
```

Out[59]:
```
array([0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1,
1])
```

In [60]:
```
list(X_test['chocolate'])
```

Out[60]:
```
[0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1]
```

6. Confusion Matrix.

It is a performance measurement for machine learning. It will be used for testing the accuracy. It is a table with predicted and actual values.

True Positive = 13 - Predicted positive and is true - predicted its chocolate and its chocolate.

False Positive = 0 - Predicted positive and its false - predicted chocolate and its not.

False Negative = 1 - Predicted negative and its false - predicted not chocolate and its not chocolate.

True Negative = 8 - Predicted negative and its true - predicted not chocolate and it is.

In [61]:
```
cnf_matrix = metrics.confusion_matrix(y_test, y_pred)
cnf_matrix
```

Out[61]:
```
array([[13,  0],
       [ 1,  8]])
```

7. Accuracy.

The result is a percentage which represents the correct predictions of the test data.

accuracy = True Positive + True Negative/ Total. The result is 95.4545%.

The mean squared error is the average squared difference between the estimated values and true value. It is used to find the perecentage error. The result is 0.04545 or 0.45%.

In [62]:
```
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))

# or

diagonal_sum = cnf_matrix.trace()
sum_of_all_elements = cnf_matrix.sum()

print("The Accuracy of the model is:", diagonal_sum/sum_of_all_elements*
100)

cost=mean_squared_error(y_test,y_pred)
print("\nThe error is {}".format(cost))
```

```
Accuracy: 0.9545454545454546
The Accuracy of the model is: 95.45454545454545

The error is 0.045454545454545456
```

8. Conclusion.

After performing machine learning we can identify that the most important feature is chocolate in candy and we can now predict if a candy is going to be chocolate based on what the other features are at 95.45454545454545% accuracy.