Exploring Nutritional Data

USING DATA SCIENCE MODELS

JORDAN MCMULLEN

Problem statement and hypothesis

- Problem Statement
  - The aim of this research is to see if by looking at the key nutritional components of food can the number of calories be predicted and if so which nutritional component has the best ability to do so.
- Hypotheses
  - Can the number of calories be predicted using if the key components of fat/protein/sugar/carbohydrate/water are known?
  - Which food component has the greatest correlation and ability to predict the calories in food?

Nutritional values for common foods and products data set

- https://www.kaggle.com/datasets/trolukovich/nutritional-values-for-common-foods-and-products/data
- 77 columns
  - Many different components of food
  - E.g. individual vitamins, proteins, sugars
  - All except the name column are continuous quantitative variables.
- 8789 rows

name	serving_size	calories	total_fat	saturated_	cholesterol	sodium
Cornstarch	100 g	381	0.1g		0	9.00 mg
Nuts, pecans	100 g	691	72g	6.2g	0	0.00 mg
Eggplant, raw	100 g	25	0.2g		0	2.00 mg
Teff, uncooked	100 g	367	2.4g	0.4g	0	12.00 mg
Sherbet, orange	100 g	144	2g	1.2g	1mg	46.00 mg
Cauliflower, raw	100 g	25	0.3g	0.1g	0	30.00 mg
Taro leaves, raw	100 g	42	0.7g	0.2g	0	3.00 mg
Lamb, raw, ground	100 g	282	23g	10g	73mg	59.00 mg
Cheese, camembert	100 g	300	24g	15g	72mg	842.00 mg
Vegetarian fillets	100 g	290	18g	2.8g	0	490.00 mg
PACE, Picante Sauce	100 g	25	0g		0	781.00 mg

Pre-processing steps

- Problem: all variables as objects and containing units

```
#removing units from each input and converting to float values for entire df

df = df.astype(str)

def remove_units_and_convert(column_name):
    df[column_name] = df[column_name].str.replace('[a-zA-Z]+', '', regex=True) # Replace any non num charater with nothing
    df[column_name] = df[column_name].replace('', '0')  # Replace empty strings with '0'
    df[column_name] = df[column_name].astype(float)

# Loop through all columns and apply the conversion
for col in df.columns:
    remove_units_and_convert(col)
```

```
#renaming some columns to include the untis in the name

df.rename(columns={'calories': 'calories_100g',
                   'protein': 'protein_g',
                   'carbohydrate': 'carbohydrate_g',
                   'fiber': 'fiber_g',
                   'water': 'water_g',
                   'sugars': 'sugars_g',
                   'fat': 'fat_g'}, inplace=True)
```

- Problem: Need to group the calorie into ranges 1 = 0-200, 2 = 201-400, 3 = 401-600, 4 = 601-800, 5 = 801-1000

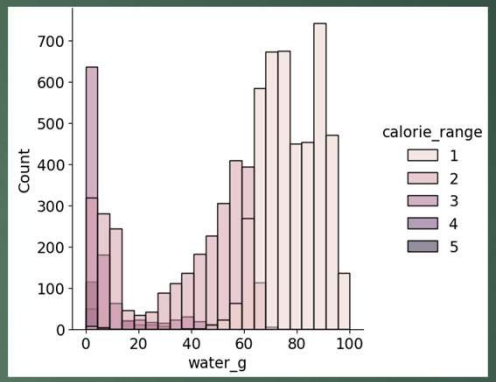
```
calorie_ranges = [0, 200, 400, 600, 800, 1000]

df['calorie_range'] = pd.cut(df['calories_100g'], bins=calorie_ranges, labels=False, right=False)

label_mapping = {0: 1, 1: 2, 2: 3, 3: 4, 4: 5}
df['calorie_range'] = df['calorie_range'].map(label_mapping)
df['calorie_range'].fillna(0, inplace=True)
df['calorie_range'] = df['calorie_range'].astype(int)
```

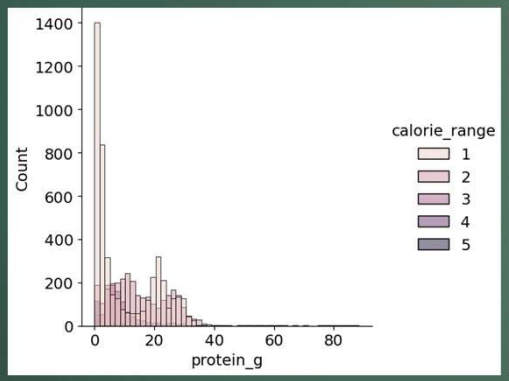
calorie_range	calories_100g
2	381
4	691
1	25
2	367
1	144

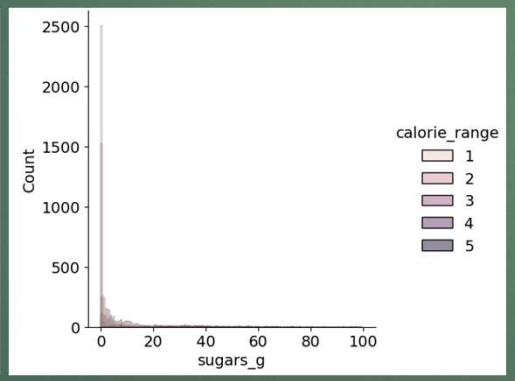
- Problem: Duplicates in the data that needed removing

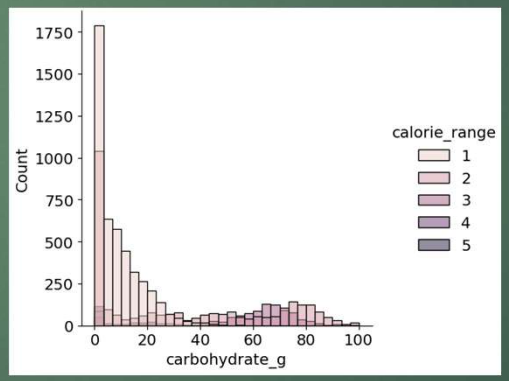
Target Features and EDA

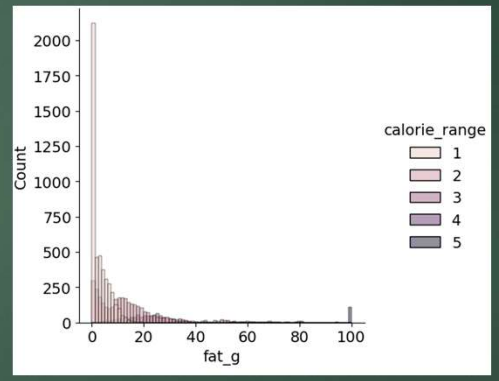
	calories_100g	calorie_range	protein_g	sugars_g	carbohydrate_g	fat_g	water_g
count	8744.000000	8744.000000	8744.000000	8744.000000	8744.000000	8744.000000	8744.000000
mean	226.081885	1.660682	11.348199	6.728935	22.083791	10.545130	54.151072
std	169.810299	0.832146	10.539750	13.698486	27.248852	15.839377	30.749150
min	0.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	91.000000	1.000000	2.380000	0.000000	0.050000	0.940000	29.975000
50%	191.000000	1.000000	8.025000	0.480000	9.285000	5.100000	63.080000
75%	336.000000	2.000000	19.900000	5.952500	34.570000	13.667500	78.032500
max	902.000000	5.000000	88.320000	99.800000	100.000000	100.000000	100.000000





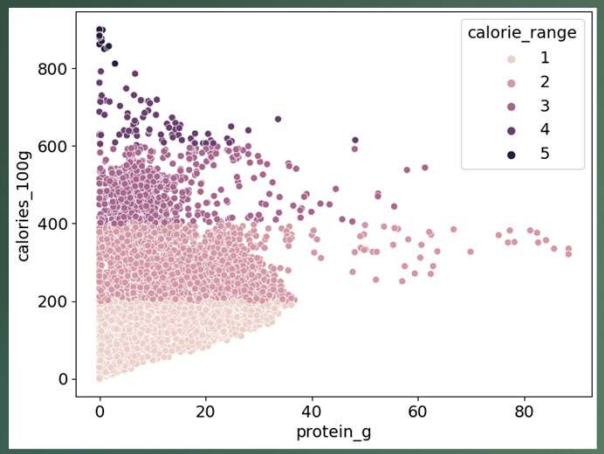


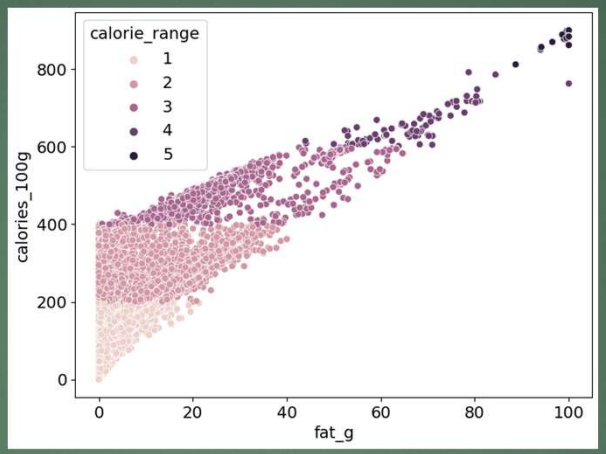


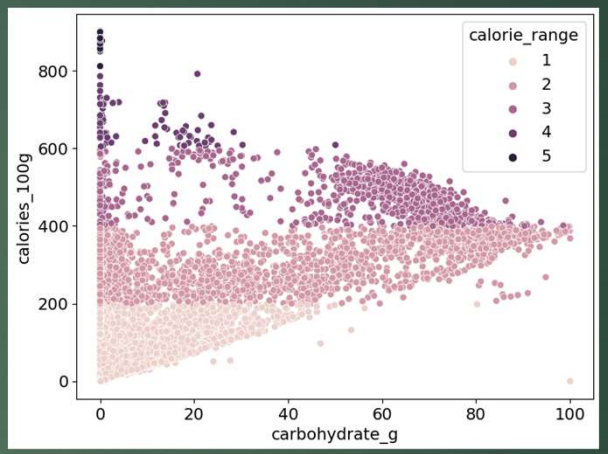


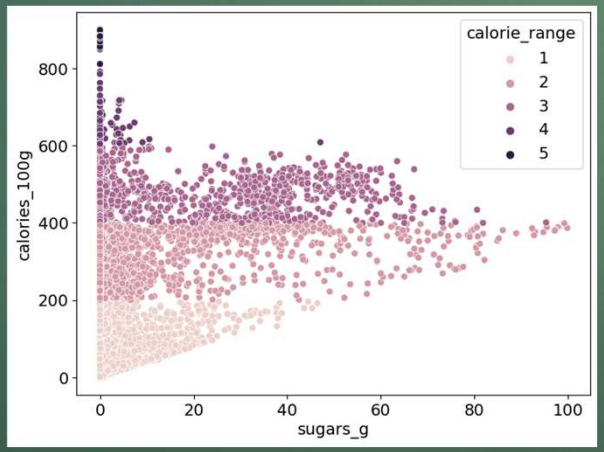
With the large number of foods that contain 0 or each component we can see the distributions are positively skewed and not normally distributed

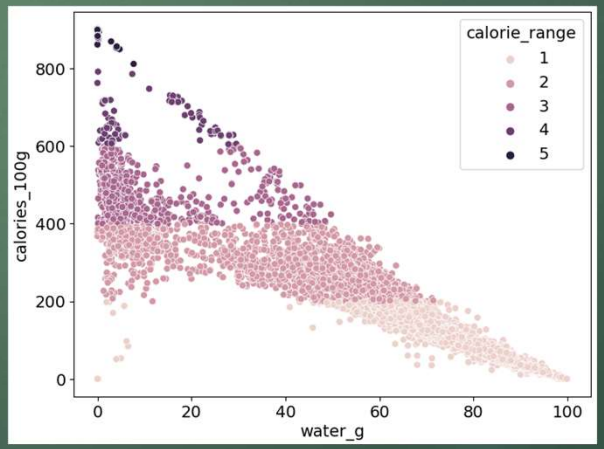
EDA cont.


calorie_range
1
2
3
4
5
calories_100g
protein_g
0
200
400
600
800
0
20
40
60
80

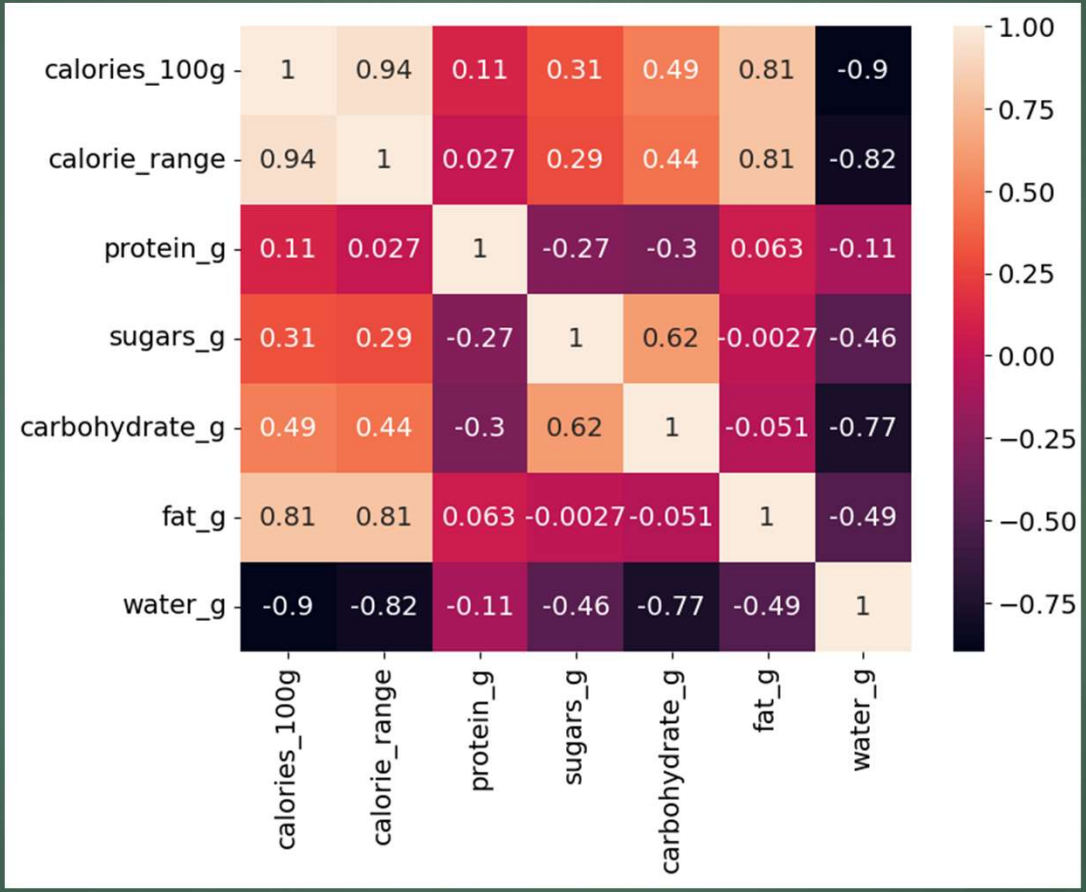

calorie_range
1
2
3
4
5
calories_100g
fat_g
0
200
400
600
800
0
20
40
60
80
100


calorie_range
1
2
3
4
5
calories_100g
carbohydrate_g
0
200
400
600
800
0
20
40
60
80
100


calorie_range
1
2
3
4
5
calories_100g
sugars_g
0
200
400
600
800
0
20
40
60
80
100


calorie_range
1
2
3
4
5
calories_100g
water_g
0
200
400
600
800
0
20
40
60
80
100

Correlation Matrix



KNN

- Using KNN resulting in an accuracy of 97 when using the 5 feature columns

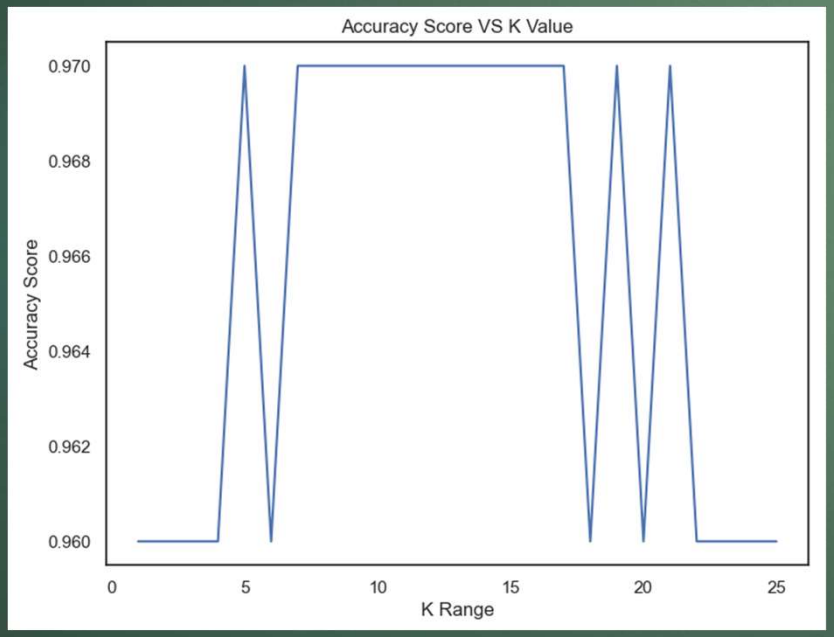
```
Accuracy with 5 features ('protein_g', 'sugars_g', 'carbohydrate_g', 'fat_g', 'water_g'): 0.97
Accuracy with 4 features ('protein_g', 'sugars_g', 'carbohydrate_g', 'fat_g'): 0.96
Accuracy with 4 features ('protein_g', 'sugars_g', 'carbohydrate_g', 'water_g'): 0.95
Accuracy with 4 features ('protein_g', 'sugars_g', 'fat_g', 'water_g'): 0.97
Accuracy with 4 features ('protein_g', 'carbohydrate_g', 'fat_g', 'water_g'): 0.97
Accuracy with 4 features ('sugars_g', 'carbohydrate_g', 'fat_g', 'water_g'): 0.97
Accuracy with 3 features ('protein_g', 'sugars_g', 'carbohydrate_g'): 0.78
Accuracy with 3 features ('protein_g', 'sugars_g', 'fat_g'): 0.88
Accuracy with 3 features ('protein_g', 'sugars_g', 'water_g'): 0.89
Accuracy with 3 features ('protein_g', 'carbohydrate_g', 'fat_g'): 0.97
Accuracy with 3 features ('protein_g', 'carbohydrate_g', 'water_g'): 0.95
Accuracy with 3 features ('protein_g', 'fat_g', 'water_g'): 0.97
Accuracy with 3 features ('sugars_g', 'carbohydrate_g', 'fat_g'): 0.93
Accuracy with 3 features ('sugars_g', 'carbohydrate_g', 'water_g'): 0.9
Accuracy with 3 features ('sugars_g', 'fat_g', 'water_g'): 0.97
Accuracy with 3 features ('carbohydrate_g', 'fat_g', 'water_g'): 0.97
Accuracy with 2 features ('protein_g', 'sugars_g'): 0.68
Accuracy with 2 features ('protein_g', 'carbohydrate_g'): 0.78
Accuracy with 2 features ('protein_g', 'fat_g'): 0.85
Accuracy with 2 features ('protein_g', 'water_g'): 0.87
Accuracy with 2 features ('sugars_g', 'carbohydrate_g'): 0.69
Accuracy with 2 features ('sugars_g', 'fat_g'): 0.8
Accuracy with 2 features ('sugars_g', 'water_g'): 0.86
Accuracy with 2 features ('carbohydrate_g', 'fat_g'): 0.93
Accuracy with 2 features ('carbohydrate_g', 'water_g'): 0.9
Accuracy with 2 features ('fat_g', 'water_g'): 0.97
Accuracy with 1 features ('protein_g',): 0.63
Accuracy with 1 features ('sugars_g',): 0.56
Accuracy with 1 features ('carbohydrate_g',): 0.67
Accuracy with 1 features ('fat_g',): 0.72
Accuracy with 1 features ('water_g',): 0.83
```

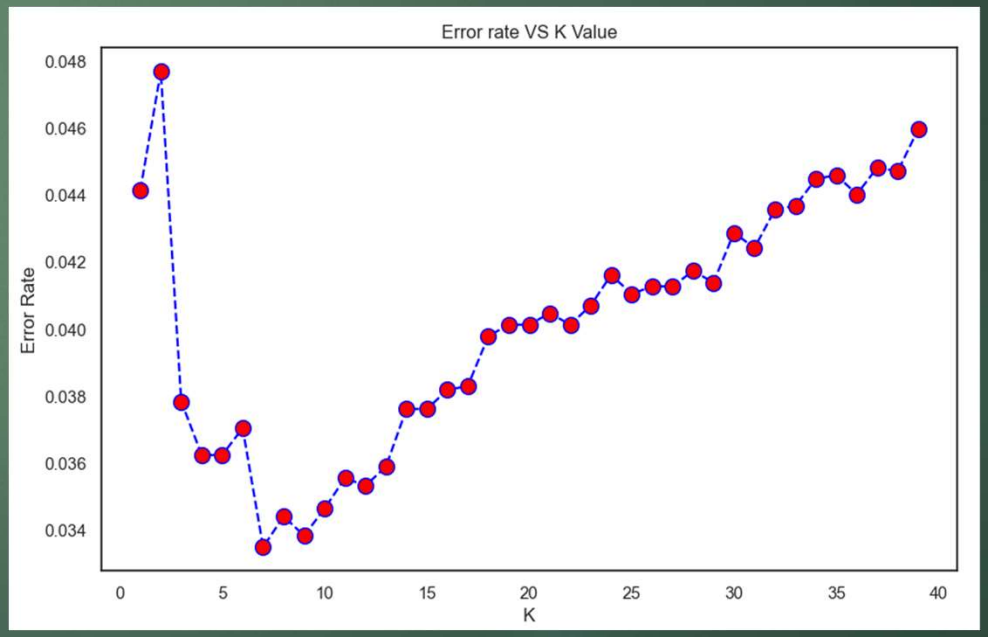
```python
def predict_from_input():
    knn = KNeighborsClassifier(n_neighbors=7)
    F = cols
    C = cal_range

    X_train,X_test,y_train,y_test=train_test_split(F,C,test_size=0.20, random_state = 2)

    knn.fit(X_train,y_train)

    user_input_values = []
    protein = input("Enter value for protien: ")
    user_input_values.append(float(protein))
    sugar = input("Enter value for sugar: ")
    user_input_values.append(float(sugar))
    carb = input("Enter value for carbohydrate: ")
    user_input_values.append(float(carb))
    fat = input("Enter value for fat: ")
    user_input_values.append(float(fat))
    water = input("Enter value for water: ")
    user_input_values.append(float(water))

    prediction = knn.predict([user_input_values])

    if prediction[0] ==1:
        print("This contains between 0-200 calories.")
    elif prediction[0] == 2:
        print("This contains between 201-400 calories.")
    elif prediction[0] == 3:
        print("This contains between 401-600 calories.")
    elif prediction[0] == 4:
        print("This contains between 601-800 calories.")
    elif prediction[0] == 5:
        print("This contains between 801-1000 calories.")
    else:
        print('This is super unhealthy')

    return prediction

predict_from_input()
```

```
Enter value for protien: 52
Enter value for sugar: 27
Enter value for carbohydrate: 12
Enter value for fat: 26
Enter value for water: 5
This contains between 401-600 calories.

array([3])
```

KNN cont.

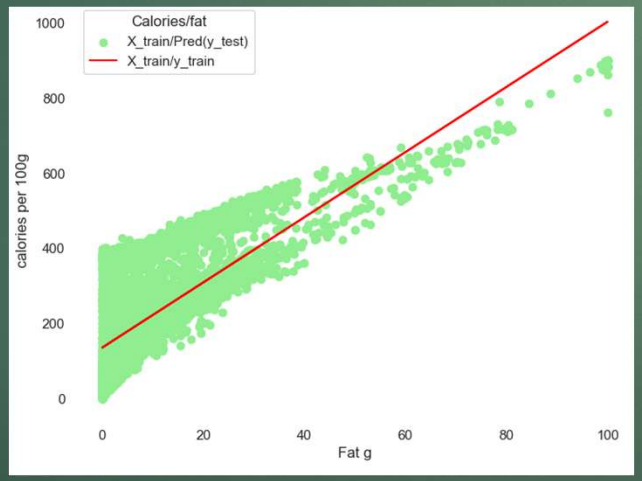
	precision	recall	f1-score	support
1	0.99	0.99	0.99	906
2	0.92	0.95	0.94	569
3	0.93	0.84	0.88	239
4	0.88	0.93	0.90	15
5	1.00	1.00	1.00	20
accuracy			0.96	1749
macro avg	0.94	0.94	0.94	1749
weighted avg	0.96	0.96	0.96	1749

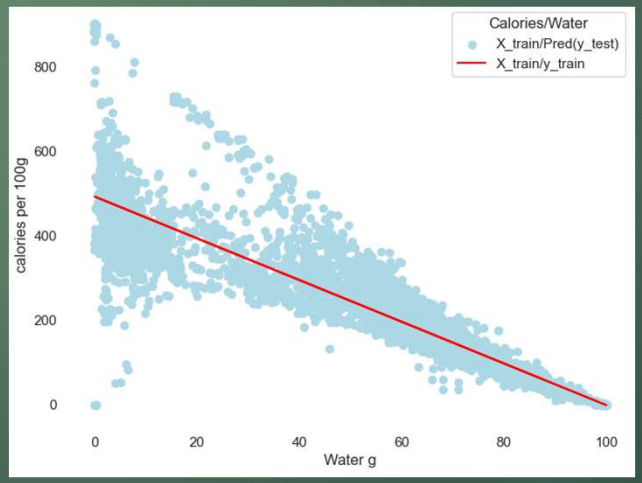




Linear Regression

Fat	Water
R-Squared 67.09%	R-Squared 80.60%
Mean Square Error 9775.75	Mean Square Error 5763.91
Mean Absolute Error 79.89	Mean Absolute Error 42.80
Intercept 135.86	Intercept 492.99
Coefficient 8.68	Coefficient -4.94

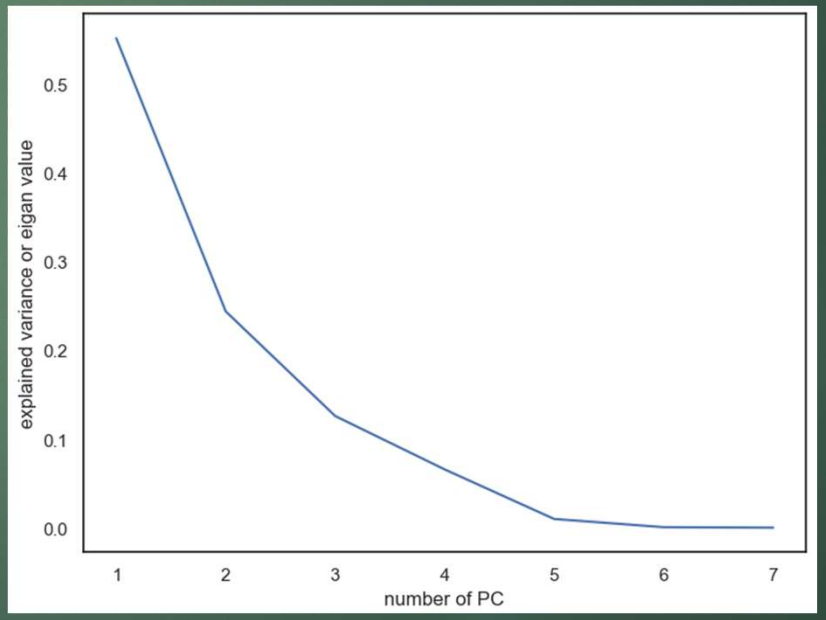




Other Models

- Logistic Regression
  - 96% accuracy
  - About 1% lower with all features then the linear regression
- PCA in linear regression
  - Found to be less accurate than linear regression with PCA

	2 PC	3 PC	5PC
R-Squared	81.89%	90.47%	90.63%
MSE	0.14	0.07	0.07
MAE	0.31	0.22	0.22



Challenges and successes

Challenges

- Lack of programming ability
- Cleaning/preparation of data
- Organization of the notebook

Successes

- Growing confidence in python/pandas
- Initial success and high accuracy of models
- Exploring model functions and tuning them to get greater results

Future research and application

Future research on this data

- Explore the data fully
- See how other nutritional component correlate to other

Personal future learning

- Time series
- NLP
- Explore and understand more and more models and code

- Application for these models
- Use within the food industry in product development
- Used to further understand the nutritional components in foods

Thank you

JORDAN MCMULLEN