# Getting the Data

In [1]:
```
import pandas as pd
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
```

In [2]:
```
data=pd.read_csv('/kaggle/input/foodcom-recipes-and-reviews/recipes.csv')
```

In [3]:
```
data.head()
```

Out[3]:

| | RecipeId | Name | AuthorId | AuthorName | CookTime | PrepTime | TotalTime | DatePublished |
|---|---|---|---|---|---|---|---|---|
| **0** | 38 | Low-Fat Berry Blue Frozen Dessert | 1533 | Dancer | PT24H | PT45M | PT24H45M | 1999-08-09T21:46:00Z |
| **1** | 39 | Biryani | 1567 | elly9812 | PT25M | PT4H | PT4H25M | 1999-08-29T13:12:00Z |
| **2** | 40 | Best Lemonade | 1566 | Stephen Little | PT5M | PT30M | PT35M | 1999-09-05T19:52:00Z |
| **3** | 41 | Carina's Tofu-Vegetable Kebabs | 1586 | Cyclopz | PT20M | PT24H | PT24H20M | 1999-09-03T14:54:00Z |
| **4** | 42 | Cabbage Soup | 1538 | Duckie067 | PT30M | PT20M | PT50M | 1999-09-19T06:19:00Z |

5 rows × 28 columns

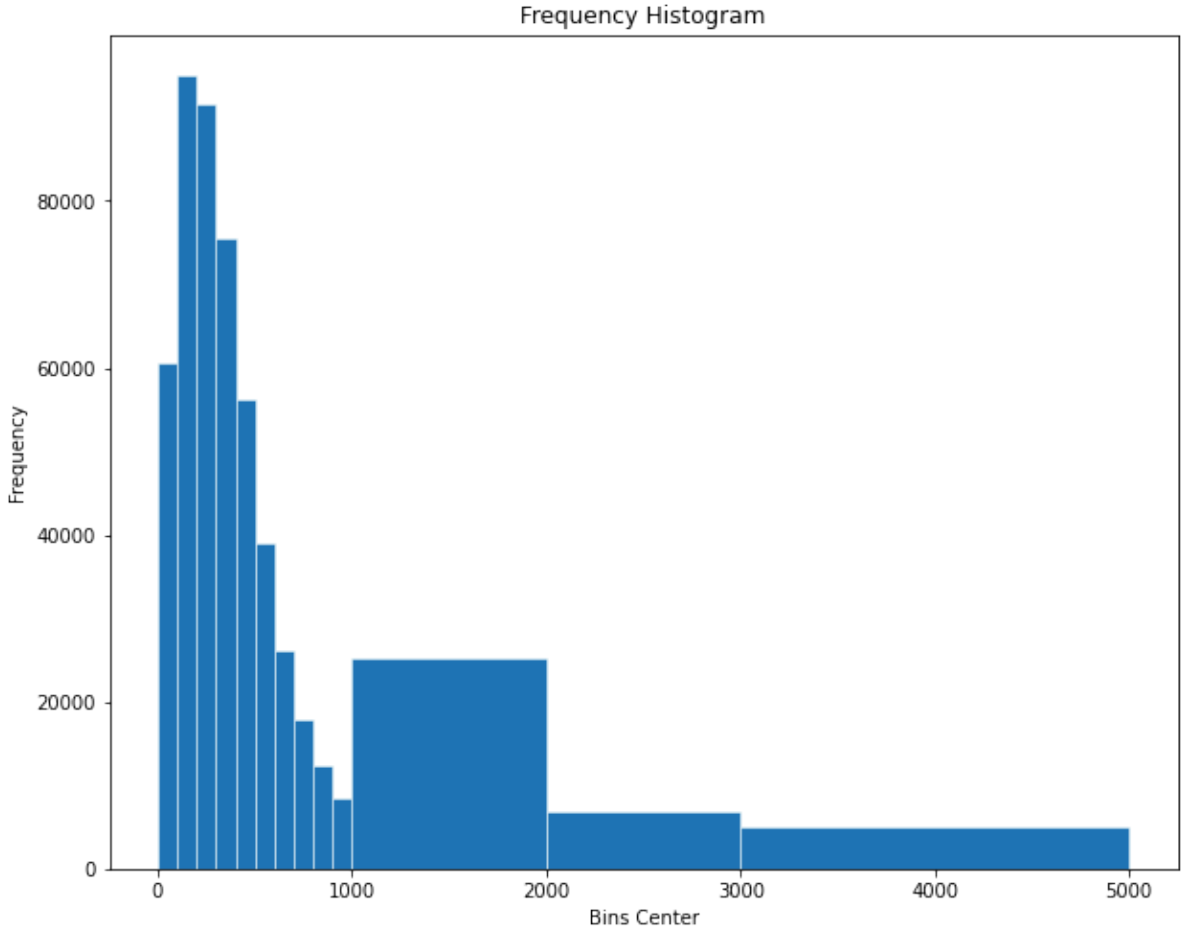
# Exploring the data

In [4]:
```
data.info()
```

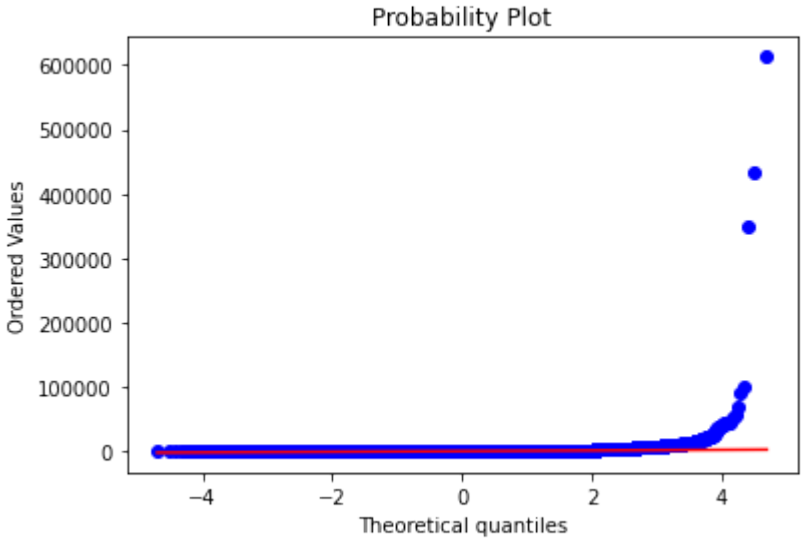
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 522517 entries, 0 to 522516
Data columns (total 28 columns):
 #   Column                      Non-Null Count   Dtype  
---  ------                      --------------   -----  
 0   RecipeId                    522517 non-null  int64  
 1   Name                        522517 non-null  object 
 2   AuthorId                    522517 non-null  int64  
 3   AuthorName                  522517 non-null  object 
 4   CookTime                    439972 non-null  object 
 5   PrepTime                    522517 non-null  object 
 6   TotalTime                   522517 non-null  object 
 7   DatePublished               522517 non-null  object 
 8   Description                 522512 non-null  object 
 9   Images                      522516 non-null  object 
 10  RecipeCategory              521766 non-null  object 
 11  Keywords                    505280 non-null  object 
 12  RecipeIngredientQuantities  522514 non-null  object 
 13  RecipeIngredientParts       522517 non-null  object 
 14  AggregatedRating            269294 non-null  float64
 15  ReviewCount                 275028 non-null  float64
 16  Calories                    522517 non-null  float64
 17  FatContent                  522517 non-null  float64
 18  SaturatedFatContent         522517 non-null  float64
 19  CholesterolContent          522517 non-null  float64
 20  SodiumContent               522517 non-null  float64
 21  CarbohydrateContent         522517 non-null  float64
 22  FiberContent                522517 non-null  float64
 23  SugarContent                522517 non-null  float64
 24  ProteinContent              522517 non-null  float64
 25  RecipeServings              339606 non-null  float64
 26  RecipeYield                 174446 non-null  object 
 27  RecipeInstructions          522517 non-null  object 
dtypes: float64(12), int64(2), object(14)
memory usage: 111.6+ MB
```

In [5]:

```
fig, ax = plt.subplots(figsize=(10, 8))
plt.title('Frequency Histogram')
plt.ylabel('Frequency')
plt.xlabel('Bins Center')
ax.hist(data.Calories.to_numpy(),bins=[0,100,200,300,400,500,600,700,800,900,1000,1
plt.show()
```




In [6]:
```
import pylab
import scipy.stats as stats
stats.probplot(data.Calories.to_numpy(), dist="norm", plot=pylab)
pylab.show()
```




# Preparing the Data

**We start by extracting the columns that we are interessted in. Since we are building a recommendation engine that takes the recipes nutritional characteristics, we start by extracting a sub data with the relevant columns. We may still need other columns for our project. However, we will mainly use the columns with nutritional information for trainning our model.**

In [7]:
```
dataset=data.copy()
columns=['RecipeId','Name','CookTime','PrepTime','TotalTime','RecipeIngredientParts
dataset=dataset[columns]
```

In [8]:
```
max_Calories=2000
max_daily_fat=100
max_daily_Saturatedfat=13
max_daily_Cholesterol=300
max_daily_Sodium=2300
max_daily_Carbohydrate=325
max_daily_Fiber=40
max_daily_Sugar=40
max_daily_Protein=200
max_list=[max_Calories,max_daily_fat,max_daily_Saturatedfat,max_daily_Cholesterol,m
```

In [9]:
```
extracted_data=dataset.copy()
for column,maximum in zip(extracted_data.columns[6:15],max_list):
    extracted_data=extracted_data[extracted_data[column]<maximum]
```

In [10]:
```
extracted_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 375703 entries, 0 to 522515
Data columns (total 16 columns):
 #   Column                 Non-Null Count   Dtype
---  ------                 --------------   -----
 0   RecipeId               375703 non-null  int64
 1   Name                   375703 non-null  object
 2   CookTime               313207 non-null  object
 3   PrepTime               375703 non-null  object
 4   TotalTime              375703 non-null  object
 5   RecipeIngredientParts  375703 non-null  object
 6   Calories               375703 non-null  float64
 7   FatContent             375703 non-null  float64
 8   SaturatedFatContent    375703 non-null  float64
 9   CholesterolContent     375703 non-null  float64
 10  SodiumContent          375703 non-null  float64
 11  CarbohydrateContent    375703 non-null  float64
 12  FiberContent           375703 non-null  float64
 13  SugarContent           375703 non-null  float64
 14  ProteinContent         375703 non-null  float64
 15  RecipeInstructions     375703 non-null  object
dtypes: float64(9), int64(1), object(6)
memory usage: 48.7+ MB
```

In [11]:
```
extracted_data.iloc[:,6:15].corr()
```

Out[11]:

| | Calories | FatContent | SaturatedFatContent | CholesterolContent | SodiumConte |
|---|---|---|---|---|---|
| **Calories** | 1.000000 | 0.767356 | 0.603317 | 0.478934 | 0.501( |
| **FatContent** | 0.767356 | 1.000000 | 0.767357 | 0.440515 | 0.381( |
| **SaturatedFatContent** | 0.603317 | 0.767357 | 1.000000 | 0.512186 | 0.319( |
| **CholesterolContent** | 0.478934 | 0.440515 | 0.512186 | 1.000000 | 0.335( |
| **SodiumContent** | 0.501082 | 0.381944 | 0.319671 | 0.335843 | 1.000( |
| **CarbohydrateContent** | 0.711640 | 0.223549 | 0.176623 | 0.066104 | 0.294( |
| **FiberContent** | 0.458711 | 0.192142 | 0.044003 | -0.047346 | 0.260( |
| **SugarContent** | 0.180895 | 0.042603 | 0.090721 | -0.036112 | -0.055( |
| **ProteinContent** | 0.689447 | 0.468088 | 0.388618 | 0.675302 | 0.500( |

In [12]:
```
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
prep_data=scaler.fit_transform(extracted_data.iloc[:,6:15].to_numpy())
```

In [13]:
```
prep_data
```

Out[13]:
```
array([[-0.55093359, -0.91281917, -0.77924852, ...,  0.15672078,
         2.35502102, -0.68338127],
       [ 1.47428542,  1.13139595, -0.0647135 , ...,  3.91055068,
         2.56324444,  1.25158691],
       [-0.92414618, -1.11248669, -1.12222533, ...,  0.4855234 ,
         0.98513013, -0.60183088],
       ...,
       [ 0.49162165,  0.73206091,  1.85024037, ..., -0.61048534,
         1.76322815, -0.56476253],
       [ 0.25704672,  0.03797856,  1.02137974, ..., -0.61048534,
         1.54404561, -0.63148557],
       [-1.40937801, -1.09347074, -1.12222533, ..., -0.82968708,
        -0.94367625, -0.74269064]])
```

# Trainning the model

## Fitting the model

In [14]:
```
from sklearn.neighbors import NearestNeighbors
neigh = NearestNeighbors(metric='cosine',algorithm='brute')
neigh.fit(prep_data)
```

Out[14]:
```
NearestNeighbors(algorithm='brute', metric='cosine')
```

In [15]:
```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import FunctionTransformer
transformer = FunctionTransformer(neigh.kneighbors,kw_args={'return_distance':False
pipeline=Pipeline([('std_scaler',scaler),('NN',transformer)])
```

In [16]:
```
params={'n_neighbors':10,'return_distance':False}
pipeline.get_params()
pipeline.set_params(NN__kw_args=params)
```

Out[16]:
```
Pipeline(steps=[('std_scaler', StandardScaler()),
                ('NN',
                 FunctionTransformer(func=<bound method KNeighborsMixin.kneighbors
of NearestNeighbors(algorithm='brute', metric='cosine')>,
                                     kw_args={'n_neighbors': 10,
                                              'return_distance': False}))])
```

In [17]:
```
pipeline.transform(extracted_data.iloc[0:1,6:15].to_numpy())[0]
```

Out[17]:
```
array([     0, 333440, 349044, 109248,  19679, 156831, 144322, 301119,
       262699, 332342])
```

# Testing the model

In [18]:
```
extracted_data.iloc[pipeline.transform(extracted_data.iloc[0:1,6:15].to_numpy())[0]
```

Out[18]:

| | RecipeId | Name | CookTime | PrepTime | TotalTime | RecipeIngredientParts | Calories | F |
|---|---|---|---|---|---|---|---|---|
| **0** | 38 | Low-Fat Berry Blue Frozen Dessert | PT24H | PT45M | PT24H45M | c("blueberries", "granulated sugar", "vanilla ... | 170.9 | |
| **463750** | 480841 | Mango Salsa | PT5M | PT10M | PT15M | c("fresh mango", "tomatoes", "sweet onion", "f... | 152.5 | |
| **485171** | 503065 | Glazed Pineapple With Cinnamon Creme Fraiche | PT10M | PT10M | PT20M | c("lime", "honey", "ground cinnamon", "ground ... | 172.5 | |
| **158110** | 165636 | Lemon Float Punch | PT120H | PT5M | PT120H5M | c("lemons", "sugar", "water", "ginger ale", "l... | 158.4 | |
| **28595** | 32172 | L & B's Concoction | PT5M | PT5M | PT10M | c("strawberry", "strawberry", "milk", "blueber... | 167.3 | |
| **224062** | 233508 | Blueberry Mango Smoothie | NaN | PT5M | PT5M | c("vanilla-flavored soymilk", "frozen blueberr... | 147.5 | |
| **206883** | 215824 | Blueberry Orange Smoothie | NaN | PT5M | PT5M | c("blueberries", "fresh blueberries") | 179.4 | |
| **419537** | 434977 | Preserved Apple Pie Filling | PT30M | PT1H | PT1H30M | c("apples", "bottled lemon juice", "sugar", "g... | 161.3 | |
| **367808** | 381181 | Tropical Twister Smoothies | NaN | PT5M | PT5M | c("fresh mango", "papaya", "fresh pineapple ch... | 190.8 | |
| **462235** | 479288 | Summer Fruit Bowl | NaN | PT1H30M | PT1H30M | c("blueberries", "granulated sugar", "kirsch",... | 155.4 | |

In [19]:
```
extracted_data[extracted_data['RecipeIngredientParts'].str.contains("egg",regex=Fal
```

Out[19]:

| | RecipeId | Name | CookTime | PrepTime | TotalTime | RecipeIngredientParts | Calories |
|---|---|---|---|---|---|---|---|
| **3** | 41 | Carina's Tofu-Vegetable Kebabs | PT20M | PT24H | PT24H20M | c("extra firm tofu", "eggplant", "zucchini", "... | 536.1 |
| **7** | 45 | Buttermilk Pie With Gingersnap Crumb Crust | PT50M | PT30M | PT1H20M | c("sugar", "margarine", "egg", "flour", "salt"... | 228.0 |
| **12** | 50 | Biscotti Di Prato | PT50M | PT20M | PT1H10M | c("flour", "sugar", "baking powder", "salt", "... | 89.4 |
| **18** | 56 | Buttermilk Pie | PT1H | PT20M | PT1H20M | c("butter", "margarine", "sugar", "flour", "eg... | 395.9 |
| **22** | 60 | Blueberry Dessert | NaN | PT35M | PT35M | c("Bisquick baking mix", "sugar", "butter", "m... | 381.1 |
| **...** | ... | ... | ... | ... | ... | ... | ... |
| **522484** | 541351 | Spinach & Mushroom Quiche with Boursin | PT1H | PT20M | PT1H20M | c("butter", "onion", "sweet pepper", "carrots"... | 197.6 |
| **522490** | 541357 | Chocolate Rum Snowballs | PT8M | PT15M | PT23M | c("rolled oats", "sweetened flaked coconut", "... | 127.8 |
| **522500** | 541367 | Thick Peanut Pancakes | PT10M | PT45M | PT55M | c("plain flour", "baking powder", "baking soda... | 712.9 |
| **522510** | 541377 | Slow-Cooker Classic Coffee Cake | PT3H | PT20M | PT3H20M | c("all-purpose flour", "brown sugar", "butter"... | 358.9 |
| **522512** | 541379 | Meg's Fresh Ginger Gingerbread | PT35M | PT1H | PT1H35M | c("fresh ginger", "unsalted butter", "dark bro... | 316.6 |

83668 rows × 16 columns

# Creating an end to end function

In [20]:

```
def scaling(dataframe):
    scaler=StandardScaler()
    prep_data=scaler.fit_transform(dataframe.iloc[:,6:15].to_numpy())
    return prep_data,scaler

def nn_predictor(prep_data):
    neigh = NearestNeighbors(metric='cosine',algorithm='brute')
    neigh.fit(prep_data)
    return neigh
```

```
def build_pipeline(neigh,scaler,params):
    transformer = FunctionTransformer(neigh.kneighbors,kw_args=params)
    pipeline=Pipeline([('std_scaler',scaler),('NN',transformer)])
    return pipeline

def extract_data(dataframe,ingredient_filter,max_nutritional_values):
    extracted_data=dataframe.copy()
    for column,maximum in zip(extracted_data.columns[6:15],max_nutritional_values):
        extracted_data=extracted_data[extracted_data[column]<maximum]
    if ingredient_filter!=None:
        for ingredient in ingredient_filter:
            extracted_data=extracted_data[extracted_data['RecipeIngredientParts'].s
    return extracted_data

def apply_pipeline(pipeline,_input,extracted_data):
    return extracted_data.iloc[pipeline.transform(_input)[0]]

def recommand(dataframe,_input,max_nutritional_values,ingredient_filter=None,params
    extracted_data=extract_data(dataframe,ingredient_filter,max_nutritional_values)
    prep_data,scaler=scaling(extracted_data)
    neigh=nn_predictor(prep_data)
    pipeline=build_pipeline(neigh,scaler,params)
    return apply_pipeline(pipeline,_input,extracted_data)
```

In [21]:
```
test_input=extracted_data.iloc[0:1,6:15].to_numpy()
recommand(dataset,test_input,max_list)
```

Out[21]:

| | RecipeId | Name | CookTime | PrepTime | TotalTime | RecipeIngredientParts | Calories | F |
|---|---|---|---|---|---|---|---|---|
| **0** | 38 | Low-Fat Berry Blue Frozen Dessert | PT24H | PT45M | PT24H45M | c("blueberries", "granulated sugar", "vanilla ... | 170.9 | |
| **463750** | 480841 | Mango Salsa | PT5M | PT10M | PT15M | c("fresh mango", "tomatoes", "sweet onion", "f... | 152.5 | |
| **485171** | 503065 | Glazed Pineapple With Cinnamon Creme Fraiche | PT10M | PT10M | PT20M | c("lime", "honey", "ground cinnamon", "ground ... | 172.5 | |
| **158110** | 165636 | Lemon Float Punch | PT120H | PT5M | PT120H5M | c("lemons", "sugar", "water", "ginger ale", "l... | 158.4 | |
| **28595** | 32172 | L & B's Concoction | PT5M | PT5M | PT10M | c("strawberry", "strawberry", "milk", "blueber... | 167.3 | |