PUBLIC HEALTHCARE AWARENESS

621121104017 : M.JOTHI

PHASE 5 PROJECT:

PROJECT : PUBLIC HEALTH AWARENESS(NUTRITION)

➢ DOCCUMENTATION AND REPORT PROJECT

◆ TABLE OF CONTENT:

ABSTRACTION:

Awareness is essential for prevention, early detection, targeted therapy and is key to ensuring effective treatment. Being aware of a disease and its symptoms means people are more likely to take preventative action, and go for screenings, tests and check-ups.

INTRODUCTION:

Nutrition is a basic human need and a prerequisite for healthy life. A proper diet is essential from very early age of life for growth, development .

AWARENESS COMPAIGNS :

- Celebrate RDN Day. Read the Full Story.
- Celebrate NDTR Day. Read the Full Story.

- 50 Ideas to Get Involved in National Nutrition Month® Read the Full Story.
- Practice Food Safety this World Health Day. Read the Full Story.
- Reduce Risk During Cancer Control Month. Read the Full Story.
- Reduce Breast Cancer Risk.

OBJECTIVES:

- To have an understanding as to how to meet our service users nutritional needs
- To have an awareness of food hygiene regulations and to be able to put them into practice
- Understand the importance of a well balanced diet, including the importance of fluids
- Understand the legislations relating to food hygiene and nutritional requirements for service users
- Understand the basics of food hygiene and the consequences if they are not adhered .

NUTRITION COMPAIGN:

Nutrition and hydration campaign is very crucial for promoting awareness and encourage healthy diet daily. Good nutrition is required for all age group and good nutrition not only focus on food but also the drinks and poor nutrition is result to many complication like malnutrition that leads to various illness. In the developing countries campaign is very important to educate the people and encourage them about a balance diet. Nutrition campaign main important goal is to prevent malnutrition by educating, counselling, measuring health status and distributing of nutritional food.

DESIGN THINKING:

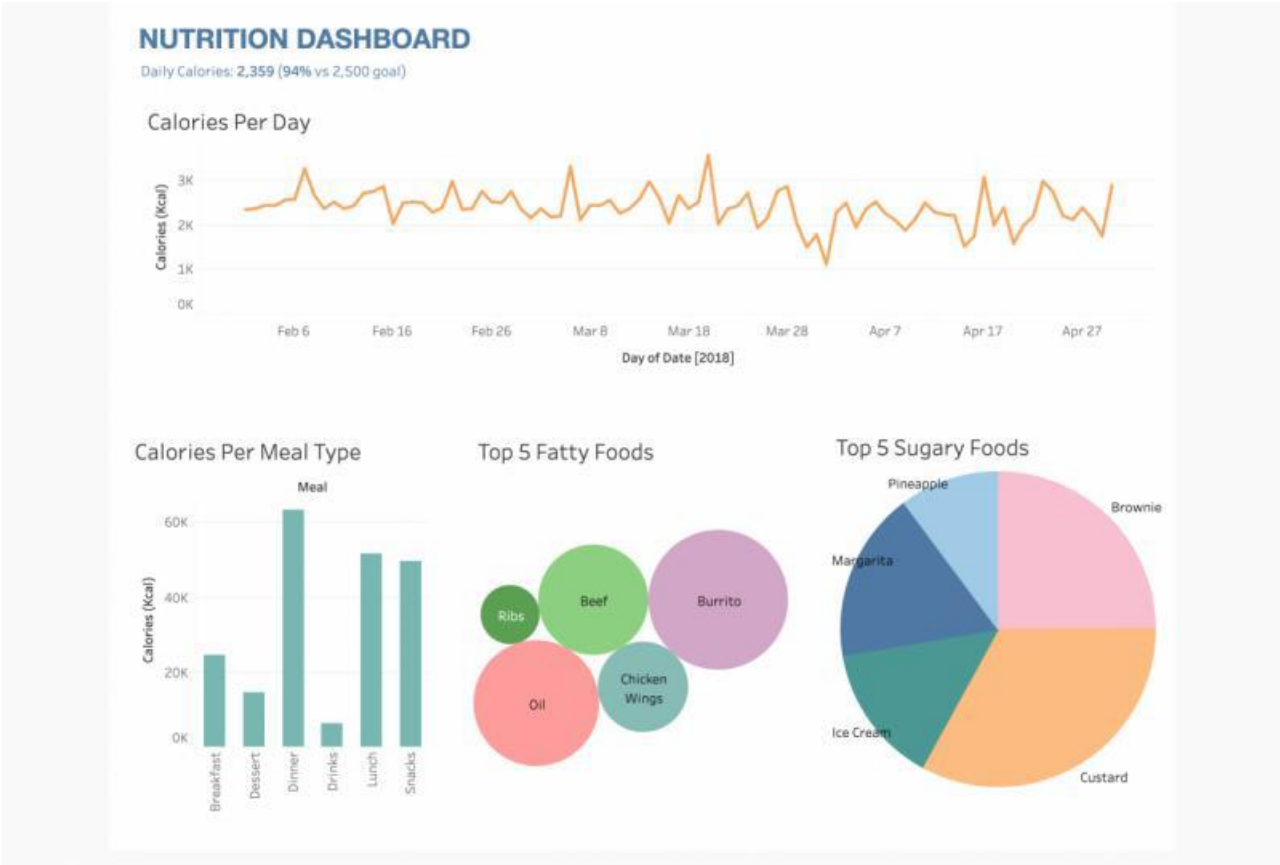
1. ANALYSIS OBJECTIVES:

- Food Nutrition Analysis helps in the detailed and perfect determination of the component nutrients present in any food item.
- Food components have vast bio metabolic roles and could affect human health severely.
- If the consumer has a clear idea about the food component, he or she may choose or reject specific food items according to his or her health condition.
- As an example, khichdi is more beneficial than plain rice or roti alone in case a person is looking for low-cost supplementary food for protein-energy malnutrition. The combined protein sourced from rice and dal makes khichdi more nutritional and wholesome.
- Nutritional Analysis and comparison of ‘Khichdi’ with various other food items helped scientists in concluding this revolutionary easy way which is now an instrumental Malnutrition.

2.DATA COLLECTION:

Nutrition surveillance in low-income countries involves the regular and systematic collection of data on nutritional outcomes and exposures, as specified in 1976 in the first guidance on the subject: “Surveillance should provide ongoing information about the nutritional conditions of the population and the factors that influence them” . Information derived from nutrition surveillance has been used in several ways: to monitor the nutrition situation; to identify factors associated with malnutrition; to inform nutrition policies and programmes; to track progress towards achieving nutrition goals; to serve as an early warning of increased nutritional risk; to assess the delivery and coverage of services; to evaluate programmes and interventions; and to detect the impact of changes in policies.

3.VISUALIZATION STRATEGY:



4.CODE INTEGRATION:

Since 2017, WFP has led a transformative agenda to establish and sustain effective integration of nutrition into multiple systems and sectors to improve diets and address the underlying causes of malnutrition. WFP has developed and applies guidance and tools that establish minimum requirements and opportunities for integrating nutrition and promoted assessment/evidence-based analysis (Fill the Nutrient Gap) and monitoring and

evaluation for nutrition integration. As a result, there is a stronger nutrition focus of WFP's delivery and enabling work. This cross-cutting approach requires that nutrition is an integral part of analysis and planning across all elements within each of the systems/sectors such as design and delivery, capacity and workforce, governance, information systems, technology and finance. WFP's contribution to nutrition outcomes through this nutrition integration approach should contribute to healthy diets and good nutrition for millions of vulnerable households.

STEPS:

1. Data Collection and Preprocessing:

- Gather historical data related to past campaigns. This data should include information on campaign attributes (e.g., target audience, messaging, channels used), engagement metrics (e.g., click-through rate, conversion rate), and other relevant factors.
- Clean and preprocess the data by handling missing values, outliers, and encoding +categorical variables. Ensure that the dataset is well-structured and ready for analysis.

2. Feature Engineering:

- Create meaningful features from the historical data that can be used as input for your machine learning model. These features could include time-based features, demographic information, and any other relevant variables.
- Consider using techniques such as one-hot encoding, feature scaling, and feature selection to improve the quality of your feature set.

3. Splitting the Data:

- Divide your dataset into two or three parts: training, validation, and testing sets. The training set is used to train the model, the validation set helps tune hyperparameters, and the testing set is used to evaluate the model's performance.

4.Model Selection:

- Choose appropriate machine learning algorithms for your predictive task. Common choices for predictive analytics in marketing include regression models (e.g., linear regression), classification models (e.g., logistic regression, decision trees, random forests), and more advanced models like gradient boosting or neural networks.

5.Model Training and Evaluation:

- Train your selected models on the training dataset and fine-tune their hyperparameters using the validation dataset. Use evaluation metrics such as accuracy, precision, recall,

F1-score, or ROC AUC (for classification tasks) or mean squared error, mean absolute error, or R-squared (for regression tasks) to assess model performance.

6.Cross-Validation:

- Consider using techniques like k-fold cross-validation to ensure that your model's performance is consistent across different subsets of your data.

7.Hyperparameter Tuning:

- Optimize your model's hyperparameters using techniques such as grid search or random search to find the best combination of parameters that yield the highest predictive accuracy.

8. Deployment:

- Once you have a well-performing model, deploy it in a production environment where it can make predictions for future campaigns.

9.Monitoring and Maintenance:

- Continuously monitor the model's performance in the production environment. Retrain the model periodically to keep it up to date with changing trends and data.

10.Interpretability:

- Consider using techniques for model interpretability, such as feature importance analysis or SHAP (SHapley Additive exPlanations), to understand which factors are driving the predictions. This can provide valuable insights for campaign optimization.

11. Feedback Loop:

- Implement a feedback loop where the model's predictions are compared to the actual campaign outcomes. Use this feedback to iteratively improve the model and campaign strategies.

PROGRAM:

```
Import numpy as np
import pandas as pd
import pandas_profiling
import warnings
warnings.filterwarnings('ignore')
import datetime
from datetime import date
```

```
import seaborn as sns
 import matplotlib.pyplot as plt
%matplotlib inline
sns.set_style("whitegrid")

import cufflinks as cf
import plotly.express as px
from plotly.offline import download_plotlyjs, init_notebook_mode, plot, iplot
init_notebook_mode(connected=True)
cf.go_offline()
import pandas_profiling
import plotly.graph_objects as go
from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV
from sklearn.metrics import accuracy_score
from sklearn.svm import SVR
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
import xgboost as xg
```

Importing the required dataset, renaming the columns, removing the commas from the columns and converting their data types

In [2]:

```
df=pd.read_csv('../input/daily-website-visitors/daily-website-visitors.csv')
df.rename(columns = {'Day.Of.Week':'day_of_week'
,'Page.Loads':'page_loads'
,'Unique.Visits':'unique_visits' ,'First.Time.Visits':'first_visits'
,'Returning.Visits':'returning_visits'}, inplace = True)
df=df.replace(',','',regex=True)
df['page_loads']=df['page_loads'].astype(int)
df['unique_visits']=df['unique_visits'].astype(int)
df['first_visits']=df['first_visits'].astype(int)
df['returning_visits']=df['returning_visits'].astype(int)
df
```

Out[2]:

Row	Day	day_of_week	Date	page_loads	unique_visits	first_visits	returning_visits
0	1	Sunday	1	9/14/2014	2146	1582	1430	152
1	2	Monday	2	9/15/2014	3621	2528	2297	231
2	3	Tuesday	3	9/16/2014	3698	2630	2352	278
3	4	Wednesday	4	9/17/2014	3667	2614	2327	287
4	5	Thursday	5	9/18/2014	3316	2366	2130	236
...	...	...	...	...	...	...	...	...
2162	2163	Saturday	7	8/15/2020	2221	1696	1373	323

2163	2164	Sunday	1	8/16/2020	2724	2037	1686	351
2164	2165	Monday	2	8/17/2020	3456	2638	2181	457

2167 rows × 8 columns

Checking for the null values if any
In [3]:
df.isna().sum()
Out[3]:
Row 0
Day 0
day_of_week 0
Date 0
page_loads 0
unique_visits 0
first_visits 0
returning_visits 0
dtype: int64
Checking for duplicate values if any
In [4]:
df.duplicated().sum()
Out[4]:
0
In [5]:
df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2167 entries, 0 to 2166
Data columns (total 8 columns):
Column Non-Null Count Dtype
--- ------ -------------- -----
0 Row 2167 non-null int64
1 Day 2167 non-null object
2 day_of_week 2167 non-null int64
3 Date 2167 non-null object
4 page_loads 2167 non-null int64
5 unique_visits 2167 non-null int64
6 first_visits 2167 non-null int64
7 returning_visits 2167 non-null int64
dtypes: int64(6), object(2)
memory usage: 135.6+ KB
generating line plot for visualizing the trend of page loads and visits over time series, it seems that page loads and visits have a constant fluctuation, means they have trend over time and are correlated to each other.
In [6]:
px.line(df,x='Date',y=['page_loads' ,'unique_visits' ,'first_visits' ,'returning_visits'],
labels={'value':'Visits'}
,title='Page Loads & visitors over Time')

9/14/201410/21/201411/27/20141/3/20152/9/20153/18/20154/24/20155/31/20157/7/20158/13/20159/19/201510/26/201512/2/20151/8/20162/14/20163/22/20164/28/20166/4/

20167/11/20168/17/20169/23/201610/30/201612/6/20161/12/20172/18/20173/27/20175/3/20176/9/20177/16/20178/22/20179/28/201711/4/201712/11/20171/17/20182/23/20184/1/20185/8/20186/14/20187/21/20188/27/201810/3/201811/9/201812/16/20181/22/20192/28/20194/6/20195/13/20196/19/20197/26/20199/1/201910/8/201911/14/201912/21/20191/27/20203/4/20204/10/20205/17/20206/23/20207/30/2020010002000300040005000600070008000
variablepage_loadsunique_visitsfirst_visitsreturning_visitsPage Loads & visitors over TimeDateVisits
This histogram plot represent the sum of unique visits for each day in the week against count of unique visits for each day in the week.
but from this plot it's hard to estimate which day had the most unique visitors, so we will explore more deeper.
In [7]:
px.histogram(df,x='unique_visits',color='Day',title='unique visits for each day')
10002000300040005000020406080100
DaySundayMondayTuesdayWednesdayThursdayFridaySaturdayunique visits for each dayunique_visitscount

With this bar plot it is clear that tuesday, wednesday, monday and thursday are the days in a week when extensive amount of traffic come to this website
In [8]:
day_imp=df.groupby(['Day'])['unique_visits'].agg(['sum']).sort_values(by='sum',ascending=False)
px.bar(day_imp,labels={'value':'sum of unique visits'},title='Sum of Unique visits for each day')
TuesdayWednesdayMondayThursdayFridaySundaySaturday00.2M0.4M0.6M0.8M1M
variablesumSum of Unique visits for each dayDaysum of unique visits
sum of unique visits for each week day over time series, we know which days get the most traffic but on what time intervals ? this graph answers to that question.
time intervals are grouped according to their relation with unique visits and days, now we can understand that in which days, months and years did the website get the most traffic.
In [9]:
px.histogram(df,x='Date',y='unique_visits',color='Day',title='Sum of unique visits for each day over Time')

9/14/20145/24/20151/31/201610/9/20166/18/20172/25/201811/4/20187/14/20193/22/202012/22/20148/31/20155/9/20161/16/20179/25/20176/4/20182/11/201910/21/20196/29/20203/31/201512/8/20158/16/20164/25/20171/2/20189/11/20185/21/20191/28/202010/29/20147/8/20153/16/201611/23/20168/2/20174/11/201812/19/20188/28/20195/6/20202/5/201510/15/20156/23/20163/2/201711/9/20177/19/20183/28/201912/5/20198/13/20205/22/20151/29/201610/7/20166/16/20172/23/201811/2/20187/12/20193/20/202012/27/20149/5/20155/14/20161/21/20179/30/20176/9/20182/16/201910/26/20197/4/2020010002000300040005000

DaySundayMondayTuesdayWednesdayThursdayFridaySaturdaySum of unique visits for each day over TimeDatesum of unique_visits
get the sum of page_loads unique_visits first_visits returning_visits related to each of their days
In [10]:

```
sums=df.groupby(['Day'])[['page_loads' ,'unique_visits' ,'first_visits' ,'returning_visits']].sum().
sort_values(
by='unique_visits',ascending=False)
sums
```

Out[10]:

	page_loads	unique_visits	first_visits	returning_visits
Day				
Tuesday	1536154	1097181	907752	189429
Wednesday	1517114	1085624	897602	188022

this grouped bar chart comes from the crosstab above and it shows the sum of page_loads, unique_visits, first_visits, returning_visits for each day

In [11]:

```
px.bar(sums,barmode='group',title='Sum of page loads and visits for each of their days')
```

TuesdayWednesdayMondayThursdayFridaySundaySaturday00.2M0.4M0.6M0.8M1M1.2M1.4M1.6M
variablepage_loadsunique_visitsfirst_visitsreturning_visitsSum of page loads and visits for each of their daysDayvalue

This is a heatmap graph that shows the correlation of each datapoint from page_loads, unique_visits, first_visits , returning_visits columns, first visits seems to have a great correlation with unique visits.

The Yellow points indicate a great correlation between first visits and unique visits, but we don't how much let's find that out

In [12]:

```
px.density_heatmap(df,
x='Date',y=['page_loads' ,'unique_visits' ,'first_visits' ,'returning_visits']
# color_continuous_scale="Viridis"
,marginal_x="histogram", marginal_y="histogram",title='Correlation for each data point')
```

9/14/201410/30/201412/15/20141/30/20153/17/20155/2/20156/17/20158/2/20159/17/201511/2/201512/18/20152/2/20163/19/20165/4/20166/19/20168/4/20169/19/201611/4/201612/20/20162/4/20173/22/20175/7/20176/22/20178/7/20179/22/201711/7/201712/23/20172/7/20183/25/20185/10/20186/25/20188/10/20189/25/201811/10/201812/26/20182/10/20193/28/20195/13/20196/28/20198/13/20199/28/201911/13/201912/29/20192/13/20203/30/20205/15/20206/30/20208/15/202002000400060008000

00.511.52countCorrelation for each data pointDatevalue

this shows the paired correlation of page_loads unique_visits first_visits returning_visits columns with annotated values we know that first visits and unique visits are correlated by 0.99 which is a great correlation and page loads have a good correlation with our target variable as well.

let's see how the correlation looks like in our next plot.

In [13]:

```
fig, ax = plt.subplots()
fig.set_size_inches(8, 6)
sns.heatmap(df[['page_loads' ,'unique_visits' ,'first_visits' ,'returning_visits']].corr(),
annot=True,
cmap='viridis_r',
fmt='g')
```

Out[13]:
<AxesSubplot:>

this scatter matrix plot shows the paired plot of page_loads unique_visits first_visits returning_visits we can see that unique visits and first visits have a straight upward line, that means that first visits are increasing as the unique visits increase. we can also other pairs and identify their level of correlation visualy.
The last thing we need is to visualize the trend line.
In [14]:
px.scatter_matrix(df[['page_loads' ,'unique_visits' ,'first_visits' ,'returning_visits']])

2000400060008000200040006000200040002000400060008000500100020004000600020004000500100
0
page_loadsunique_visitsfirst_visitsreturning_visitspage_loadsunique_visitsfirst_visitsreturning_visits Okay now we have the regression line pointing upward which confirms the trend between these two columns
In [15]:
px.scatter(
df, x='first_visits', y='unique_visits',opacity=0.4,
trendline='ols', trendline_color_override='purple',title="Regression line for unique visits and first visits"
)
500100015002000250030003500400045001000200030004000 5000
Regression line for unique visits and first visitsfirst_visitsunique_visits
there are no outliears that need to be dealt with, data is tightly packed with no dispersion except for returning visits, this column was also less correlated with our target variable.
In [16]:
px.violin(df,y=['page_loads' ,'unique_visits' ,'first_visits' ,'returning_visits'],box=True,points='all')
page_loadsunique_visitsfirst_visitsreturning_visits0100020003000400050006000700080009000

variablevalue
starting the feature engineering.
we only need these columns
In [17]:
pred_df=df[['page_loads' ,'unique_visits' ,'first_visits' ,'returning_visits','Day']]
Tuesday, wednesday, thursday and monday are the days when our website received the most traffic so we will create a feature days_f of them 1 value will define their existence and 0 will define the rest of the days.
In [18]:

```
pred_df['days_f']=np.where((df['Day']=='Tuesday') |
(df['Day']=='Wednesday') |
(df['Day']=='Thursday') |
(df['Day']=='Monday'),1,0)
pred_df
```

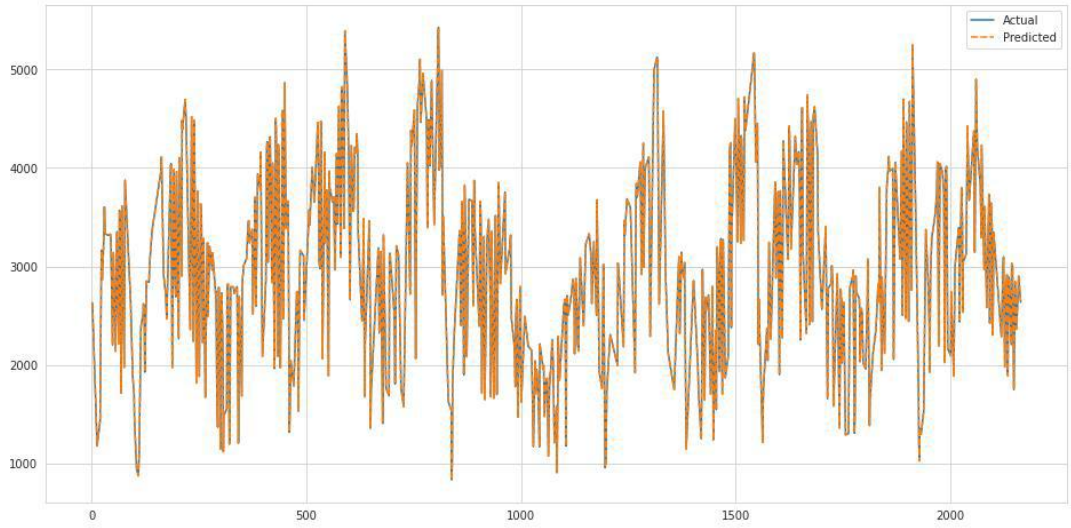
Out[18]:

page_loads	unique_visits	first_visits	returning_visits	Day	days_f
0	2146	1582	1430	152	Sunday	0
1	3621	2528	2297	231	Monday	1
2	3698	2630	2352	278	Tuesday	1
3	3667	2614	2327	287	Wednesday	1
4	3316	2366	2130	236	Thursday	1
...	...	...	...	...	...	...
2162	2221	1696	1373	323	Saturday	0
2163	2724	2037	1686	351	Sunday	0
2164	3456	2638	2181	457	Monday	1
2165	3581	2683	2184	499	Tuesday	1
2166	2064	1564	1297	267	Wednesday	1

visualize the actual and predicted values
In [26]:

```
plt.figure(figsize=(16,8))
sns.lineplot(data=lr2)
```

Out[26]:
<AxesSubplot:>

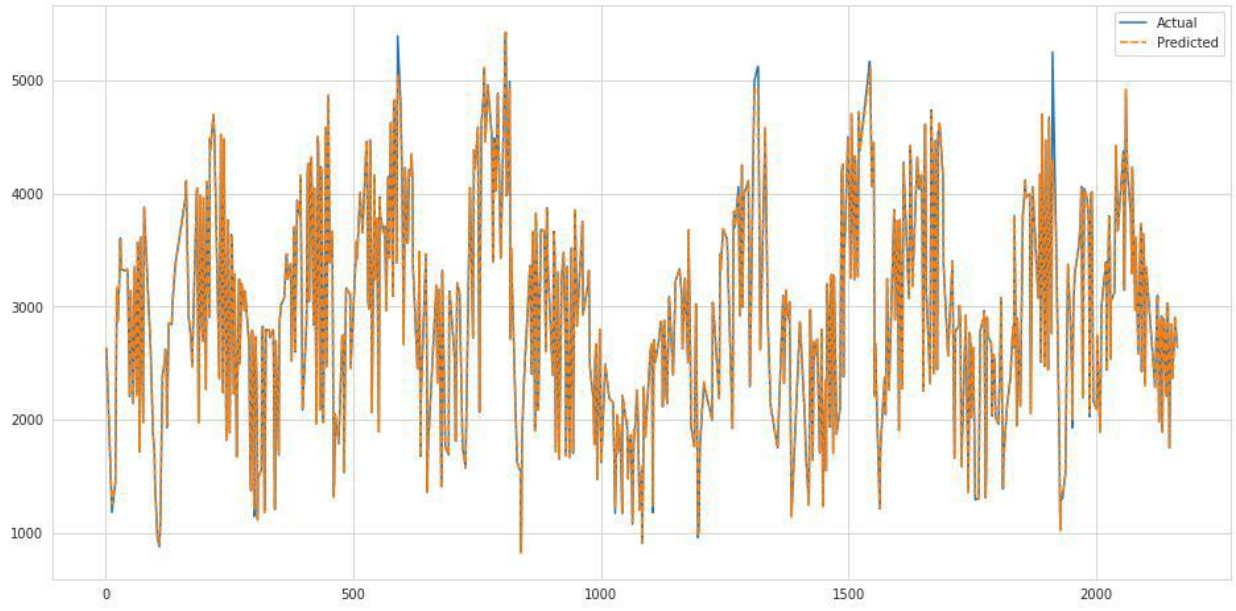


get the accuacy score of the model.
In [27]:
regressor2.score(X_test,y_test)*100
Out[27]:
100.0
Support Vector Regression In [28]:
svr_rbf = SVR(kernel='rbf', C=1e3, gamma=0.00001)
svr_rbf.fit(X_train, y_train)
Out[28]:
SVR(C=1000.0, gamma=1e-05)
In [29]:
y_pred3 = svr_rbf.predict(X_test)
In [30]:
svr = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred3})
svr

Out[30]: Predicted Actual

1486	4173	4173.783532
1602	1902	1904.847560
1460	2870	2870.181094
1134	2142	2142.904123
1513	4329	4328.316673
...	...	...
439	2579	2578.897313
271	2494	2493.887467
244	1818	1816.9

		32763
1159	3332	3331.902324
1701	2565	2564.972314



In [42]:
Xgb r.score(X_test,y_test)*100
Out[42]:
98.7655882096893

Loading and preprocessing a dataset for machine learning in the field of nutrition involves several steps. This process ensures that the data is ready for analysis and model training. Below are the key steps involved:

LOADING THE DATASET:

you can use libraries like pandas to load nutrition data from a csv file. suppose your dataset is stored in a file called 'nutrition_data.csv.'

```
import pandas as pd
data = pd.read_csv('nutrition_data.csv')
```

DATA COLLECTION:

Gather the data relevant to your nutrition analysis. This data can come from various sources, including surveys, experiments, or publicly available datasets.

DATA CLEANING:

- Remove duplicates: Check for and remove any duplicate records.

```
data = data.drop_duplicates()
```

- Handle missing values: Decide how to handle missing data, either by imputing values or removing rows with missing values.

```
data = data.dropna()  # Removes rows with missing values
```

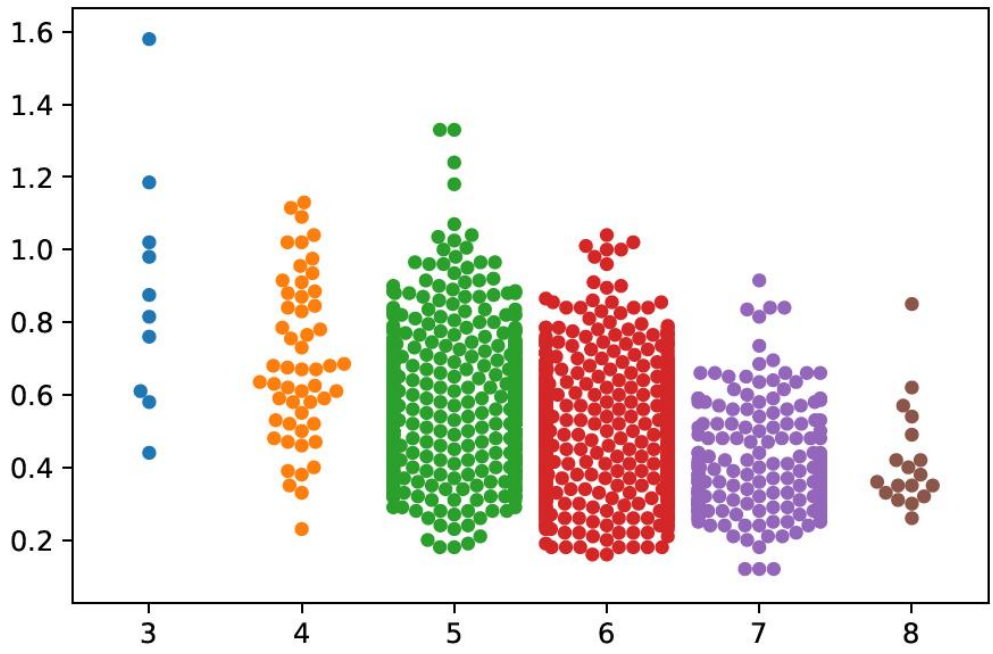
DATA FORMAT:

- Ensure that data is in a consistent format, e.g., date format, numerical values, and categorical variables.
- Outlier detection: Identify and handle outliers, as they can skew the results.

DATA EXPLORATION:

Perform exploratory data analysis (EDA) to understand the data's characteristics and distribution. This helps in identifying trends, patterns, and potential features of interest.

```
import seaborn as sns
import matplotlib.pyplot as plt
sns.pairplot(data)
```



VISUALIZATIONS:

Create plots and graphs to visualize data distributions and relationships.

FEATURE ENGINEERING:

- Select relevant features: Decide which features are essential for your nutrition analysis. These can include nutrient values, food types, meal times, and more.
- Encoding categorical variables: Convert categorical variables into numerical format using techniques like one-hot encoding or label encoding.
- Feature scaling: Scale or normalize numerical features to bring them to a common scale, which is often required for machine learning algorithms.

```
selected_features = ['calories', 'protein', 'carbs', 'fat']

data = data[selected_features]

data = pd.get_dummies(data, columns=['food_type'])
```

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

data[selected_features] = scaler.fit_transform(data[selected_features])
```

DATA SPLITTING:

Split the dataset into training, validation, and test sets. A common split might be 70% for training, 15% for validation, and 15% for testing.

```
from sklearn.model_selection import train_test_split

X = data.drop('target_column', axis=1)

y = data['target_column']
```

```
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.3,
random_state=42)

X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5,
random_state=42)
```

MODEL-SPECIFIC PREPROCESSING:

Depending on the machine learning model you plan to use, you may need to perform specific preprocessing steps. For example, text data may require tokenization, and image data may need resizing and normalization.

DATA TRANSFORMATION:

If your dataset contains time series data or sequences, you may need to apply transformations like rolling averages, windowing, or Fourier transforms.

NORMALIZATION:

In the context of nutrition, it might be essential to normalize nutrient values based on serving size or calories to make them comparable across different foods.

FEATURE SELECTION:

Identify the most important features for your specific nutrition analysis. You can use techniques like feature selection algorithms or correlation analysis.

DATA PIPELINE:

Create a data processing pipeline that includes all the preprocessing steps. This ensures consistency and reusability of the processing steps.

SAVE PREPROCESSED DATA:

Save the preprocessed data into a format suitable for machine learning, such as CSV or a database.

```
data.to_csv('preprocessed_nutrition_data.csv', index=False)
```

MACHINE LEARNING MODEL:

- Choose an appropriate machine learning model for your nutrition analysis, such as regression, classification, or deep learning.
- Train and evaluate the model using the training and validation datasets.

```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(X_train, y_train)
```

EVALUATION:

Evaluate the model's performance using appropriate metrics, such as Mean Absolute Error (MAE) for regression tasks or accuracy for classification tasks.

```
from sklearn.metrics import mean_absolute_error

y_pred = model.predict(X_val)
mae = mean_absolute_error(y_val, y_pred)
print(f'Mean Absolute Error: {mae}')
```

FINE-TUNING:

If necessary, fine-tune the model parameters to improve its performance.

TESTING:

Finally, evaluate the model on the test dataset to ensure that it generalizes well to unseen data.

INTERPRETABILITY:

- In nutrition analysis, it's often crucial to interpret the model's results. Use techniques like feature importance analysis to understand which factors influence nutrition outcomes.

- Remember that the specific steps and tools you use may vary depending on the nature of your nutrition dataset and the goals of your analysis. Additionally, ethical considerations are important when working with nutrition data, as privacy and responsposible data handling are permanent.

PROGRAM:

LOAD DATA:

The original data are provided in a csv file. It is loaded and stored in obesity_data. The first five rows are displayed below.

In [5]:
obesity_data = pd.read_csv("../input/obesity-levels/ObesityDataSet_raw_and_data_sinthetic.csv")

In [6]:
obesity_data.head()

Out[6]:

	Gender	Age	Height	Weight	family_history_with_overweight	FAVC	FCVC	NCP	CAEC	SMOKE	CH2O	SCC	FAF	TUE	CALC	MTRANS	NObeyesdad
0	Female	21.0	1.62	64.0	yes	no	2.0	3.0	Sometimes	no	2.0	no	0.0	1.0	no	Public_Transportation	Normal_Weight
1	Female	21.0	1.52	56.0	yes	no	3.0	3.0	Sometimes	yes	3.0	yes	3.0	0.0	Sometimes	Public_Transportation	Normal_Weight
2	Male	23.0	1.80	77.0	yes	no	2.0	3.0	Sometimes	no	2.0	no	2.0	1.0	Frequently	Public_Transportation	Normal_Weight
3	Male	27.0	1.80	87.0	no	no	3.0	3.0	Sometimes	no	2.0	no	2.0	0.0	Frequently	Walking	Overweight_Level_I
4	Male	22.0	1.78	89.8	no	no	2.0	1.0	Sometimes								

EXPLORATORY DATA ANALYSIS:

The output below shows that there are not missing values in the DataFrame; half of the features hold numeric (float64) values, and the other half - categorical ones. All are further explored in this Chapter. In general, the dataset is tidy, hence data cleaning was not neccessary.

```
obesity_data.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2111 entries, 0 to 2110
Data columns (total 17 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   Gender                          2111 non-null   object
 1   Age                             2111 non-null   float64
 2   Height                          2111 non-null   float64
 3   Weight                          2111 non-null   float64
 4   family_history_with_overweight  2111 non-null   object
 5   FAVC                            2111 non-null   object
 6   FCVC                            2111 non-null   float64
 7   NCP                             2111 non-null   float64
 8   CAEC                            2111 non-null   object
 9   SMOKE                           2111 non-null   object
 10  CH2O                            2111 non-null   float64
 11  SCC                             2111 non-null   object
 12  FAF                             2111 non-null   float64
 13  TUE                             2111 non-null   float64
 14  CALC                            2111 non-null   object
 15  MTRANS                          2111 non-null   object
 16  NObeyesdad                      2111 non-null   object
dtypes: float64(8), object(9)
memory usage: 280.5+ KB
```

GENDER

There are almost an equal number of females and males in the dataset. Data is available for slightly more men than women but this does not make it imbalanced.

```
count_values(obesity_data, "Gender")
```

AGE

Computing and visualizing distribution of continuous values is wrapped in a function, too. It displays not only data distribution but also its mean and median.

```
def plot_distribution(dataset, feature):
```

Function: Computes and displays distribution of features with continuous values; plots their mean and median.

Parameters: Dataset and feature with continuous values.

```
plt.hist(dataset[feature], bins = "fd")

plt.axvline(dataset[feature].mean(), color = "red", label = "mean")
plt.axvline(dataset[feature].median(), color = "orange", label = "median")

plt.xlabel(f"{feature}")
plt.ylabel("Count")
plt.legend()
plt.title(f"Distribution of values in {feature}")
plt.show()
```

The youngest person in the dataset is 14 years old, and the oldest one - 61 years of age. Values in this column are not normally distributed; the historgram is positively skewed with mean (24.31) and median (22.78) closer to the lower bound.

```
obesity_data["Age"].describe()
count    2111.000000
mean       24.312600
std         6.345968
min        14.000000
25%        19.947192
50%        22.777890
75%        26.000000
max        61.000000
Name: Age, dtype: float64
obesity_data["Age"].median()
22.77789
plot_distribution(obesity_data, "Age")
```

HEIGHT:

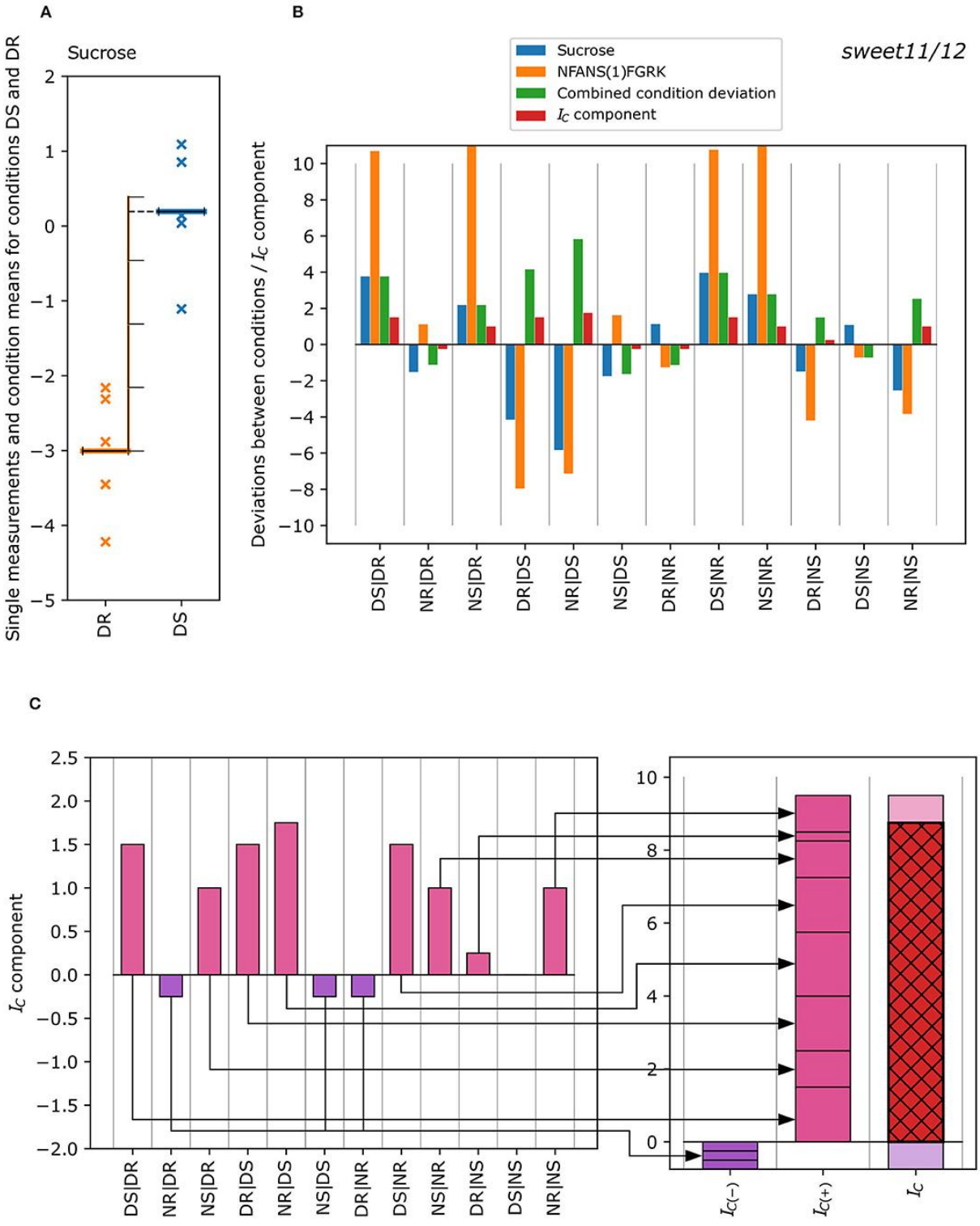
Obesity is determined by computing the Body mass index. It is a function of person's height and weight. The exact formula is

B
o
d
y
m
a
s
s
i
n

d
e
x
=
W
e
i
g
h
t
H
e
i
g
h
t
*
H
e
i
g
h
t
. Thus, height is an important element for determining obesity.

Distribution of height values is plotted below. Most people are 1.60 m - 1.85 m tall. Both mean and median values are around 1.70. Still, height values do not seem to be normally distributed.

plot_distribution(obesity_data, "Height")


A
Sucrose
Single measurements and condition means for conditions DS and DR
DR
DS
B
sweet11/12
Sucrose
NFANS(1)FGRK
Combined condition deviation
I_C component
Deviations between conditions / I_C component
DS|DR
NR|DR
NS|DR
DR|DS
NR|DS
NS|DS
DR|NR
DS|NR
NS|NR
DR|NS
DS|NS
NR|NS
C
I_C component
$I_{C(-)}$
$I_{C(+)}$
I_C

TOPIC: Continue nutrition prediction model bY feature engineering, model training, and evaluation

FEATURE ENGINEERING:

Feature engineering in nutrition involves the process of selecting, transforming, and creating relevant features or variables from nutritional data to improve the performance of machine learning models, gain insights, and make better nutritional assessments. This can be particularly useful for tasks like dietary pattern analysis, food recommendation systems, or predicting health outcomes based on nutrition.

Here are some key aspects of feature engineering in nutrition:

1. **Nutrient Aggregation**: Instead of using individual nutrient values, you might create new features that represent overall nutrient profiles. For example, you can calculate the macronutrient ratio (e.g., the percentage of calories from carbs, fats, and proteins) or the nutrient density (nutrient content per 100g of food).

2. **Food Grouping**: Categorize foods into groups like fruits, vegetables, grains, proteins, and dairy. You can then create binary or numerical features to represent food group consumption. This can help identify dietary patterns.

3. **Meal or Snack Identification**: Identify meals and snacks in dietary data. Create features that indicate whether a record corresponds to a main meal, a snack, or other eating occasion.

4. **Dietary Patterns**: Use techniques like principal component analysis (PCA) or factor analysis to create composite features that represent dietary patterns. This can help identify overarching eating habits.

5. **Portion Size Standardization**: Normalize portion sizes to a common measurement unit (e.g., grams) to make them comparable across different foods. This can involve using food portion size databases or reference guides.

6. **Nutrient Ratios**: Create features that represent nutrient ratios, such as omega-6 to omega-3 fatty acid ratio, calcium to magnesium ratio, or sodium to potassium ratio. These ratios can be more informative than individual nutrient values.

7. **Cooking Methods**: Include features that describe how foods are prepared (e.g., fried, baked, grilled). Different cooking methods can affect the nutritional content of foods.

8. **Nutritional Indices**: Calculate and include nutritional indices such as the Nutrient Rich Foods (NRF) index or the Healthy Eating Index (HEI) as features. These indices provide a summary of the overall nutritional quality of a diet.

9. **Temporal Features**: If working with time-series nutritional data, you can create features related to eating patterns over time, such as meal frequency, eating intervals, or changes in nutrient intake over weeks or months.

10. **Bioavailability**: Adjust nutrient values based on the bioavailability of nutrients in different foods. Some nutrients are more readily absorbed from certain foods, and accounting for this can improve the accuracy of nutritional assessments.

11. **Food Pairing**: Analyze which foods are commonly consumed together and create features that represent food combinations. This can help in understanding cultural dietary habits.

12. **Dietary Diversity**: Create features that measure the diversity of foods in a diet. A diverse diet is often associated with better health outcomes.

13. **Social and Cultural Context**: Include features related to cultural or social factors that influence dietary choices, such as dietary restrictions, food preferences, or eating habits specific to certain groups.

Feature engineering in nutrition should be driven by the specific goals of your analysis or model. It aims to provide relevant and meaningful information from nutritional data to better understand dietary habits, assess nutritional quality, or make personalized dietary recommendations.

PROGRAM:

```
import pandas as pd
from sklearn.preprocessing import StandardScaler
```

```
# Sample nutrition dataset
data = {
'Food': ['Apple', 'Banana', 'Salmon', 'Pasta', 'Spinach'],
'Calories': [52, 105, 206, 131, 23],
'Protein (g)': [0.26, 1.29, 22, 4.5, 2.9],
'Carbohydrates (g)': [14, 27, 0, 25, 3.6],
'Fat (g)': [0.17, 0.39, 13, 1.1, 0.4]
}
df = pd.DataFrame(data)
# Feature engineering
# 1. Nutrient density (nutrient content per 100g) df['Protein Density (g/100g)'] =
df['Protein (g)'] / (df['Calories'] / 100)
df['Carb Density (g/100g)'] = df['Carbohydrates (g)'] / (df['Calories'] / 100)
df['Fat Density (g/100g)'] = df['Fat (g)'] / (df['Calories'] / 100)
# 2. Food group (categorization)
food_groups = {
'Fruits': ['Apple', 'Banana'],
'Proteins': ['Salmon'],
'Grains': ['Pasta'],
'Vegetables': ['Spinach']
}
df['Food Group'] = df['Food'].apply(lambda x: next((group for group, foods in
food_groups.items() if x in foods), 'Other'))
# 3. Standardize nutrient values
scaler = StandardScaler()
df[['Protein (g)', 'Carbohydrates (g)', 'Fat (g)']] = scaler.fit_transform(df[['Protein (g)',
'Carbohydrates (g)', 'Fat (g)']) # 4. Calculate nutrient ratios
df['Protein to Carb Ratio'] = df['Protein (g)'] / df['Carbohydrates (g)']
df['Fat to Protein Ratio'] = df['Fat (g)'] / df['Protein (g)']
print(df)
```

MODEL TRAINING:

Training machine learning models in the field of nutrition can be highly valuable for tasks like dietary pattern analysis, personalized nutrition recommendations, or predicting health outcomes based on nutritional data. Here is a general outline of the steps involved in model training in nutrition:

- Data Collection: Gather a diverse and representative dataset of nutritional information. This may include data on food items, their nutrient compositions, portion sizes, and dietary habits of individuals.

- Data Preprocessing: Data Cleaning: Remove missing or erroneous data.

- Data Integration: Combine different data sources if necessary.

- Feature Engineering: Create relevant features from the nutritional data, as discussed in the previous response.

- Data Split: Divide the dataset into training, validation, and test sets.

- Choose a machine learning model that suits your specific task. This can be a regression model for nutrient prediction, a classification model for dietary pattern analysis, or even deep learning models for complex tasks.

- Scale or normalize the features to ensure that different features are on similar scales. This can improve the training process for many machine learning algorithms.

- Fit the chosen model to the training data. The model learns to make predictions or classifications based on the input features.

- Fine-tune the model's hyperparameters to optimize its performance on the validation set. This may involve adjusting learning rates, the number of hidden layers, or other parameters specific to the chosen model.

MODEL EVALUTION:

Model evaluation in nutrition, as in any other field, is crucial to assess how well a machine learning model or predictive algorithm is performing. The choice of evaluation metrics depends on the specific nutrition-related task you are working on. Here are some common evaluation metrics and considerations for different nutrition-related tasks:

1.Nutrient Prediction:

- **Mean Squared Error (MSE)**: This measures the average squared difference between the model's lower MSE indic8ates better performance.
- **Root Mean Squared Error (RMSE)**: RMSE is the square root of MSE and is in the same units as the nutrient being predicted. It provides a more interpretable measure of error.

2.Dietary pattern Analysis:

- **Classification Metrics**: If you're classifying dietary patterns (e.g., healthy vs. unhealthy), use metrics like accuracy, precision, recall, F1-score, and area under the receiver operating characteristic curve (AUC-ROC) depending on the balance of your classes.
- **Kappa Statistic**: This metric accounts for the possibility of agreement occurring by chance and is useful for multi-class classification tasks.

3.Food Recommendation System:

- **Top-N Recommendation Metrics**: Metrics like Precision at K, Recall at K, and Mean Average Precision (MAP) can be used to evaluate the quality of food recommendations.
- **NDCG (Normalized Discounted Cumulative Gain)**: NDCG measures the ranking quality of recommended items and takes into account the position of relevant items in the list.

4.Health Outcome Prediction:

- **Area Under the Curve (AUC)**: AUC measures the model's ability to distinguish between individuals who develop a specific health outcome and those who do not. It's commonly used for risk prediction models.
- **Concordance Index (C-index)**: Similar to AUC, the C-index assesses the discriminative ability of the model for survival or time-to-event analysis.

5.Interpretability Metrics:

Depending on the nature of the nutrition model, you might need to use metrics that assess the interpretability of the model's output. For example, the feature importance of decision trees or the relevance of features in neural networks .

PROGRAM:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
# Sample nutrition dataset
data = {
'Food': ['Apple', 'Banana', 'Salmon', 'Pasta', 'Spinach'],
'Calories': [52, 105, 206, 131, 23],
'Protein (g)': [0.26, 1.29, 22, 4.5, 2.9],
'Carbohydrates (g)': [14, 27, 0, 25, 3.6],
'Fat (g)': [0.17, 0.39, 13, 1.1, 0.4]
}
df = pd.DataFrame(data)
# Feature engineering and data preparation
# Split the data into training and testing sets
X = df[['Calories', 'Carbohydrates (g)', 'Fat (g)']]
y = df['Protein (g)'] # Predicting protein content
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
# Model training
model = LinearRegression()
model.fit(X_train, y_train)
# Model evaluation
y_pred = model.predict(X_test)
# Calculate evaluation metrics
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f"Mean Squared Error (MSE): {mse:.2f}")
print(f"R-squared (R2): {r2:.2f
```

CONCLUSION:

Public health is a vital function that requires broad public concern and support in order to fulfill society's interest in assuring the conditions in which people can be healthy. History teaches us that organized community effort to prevent disease and promote health is both valuable and effective.