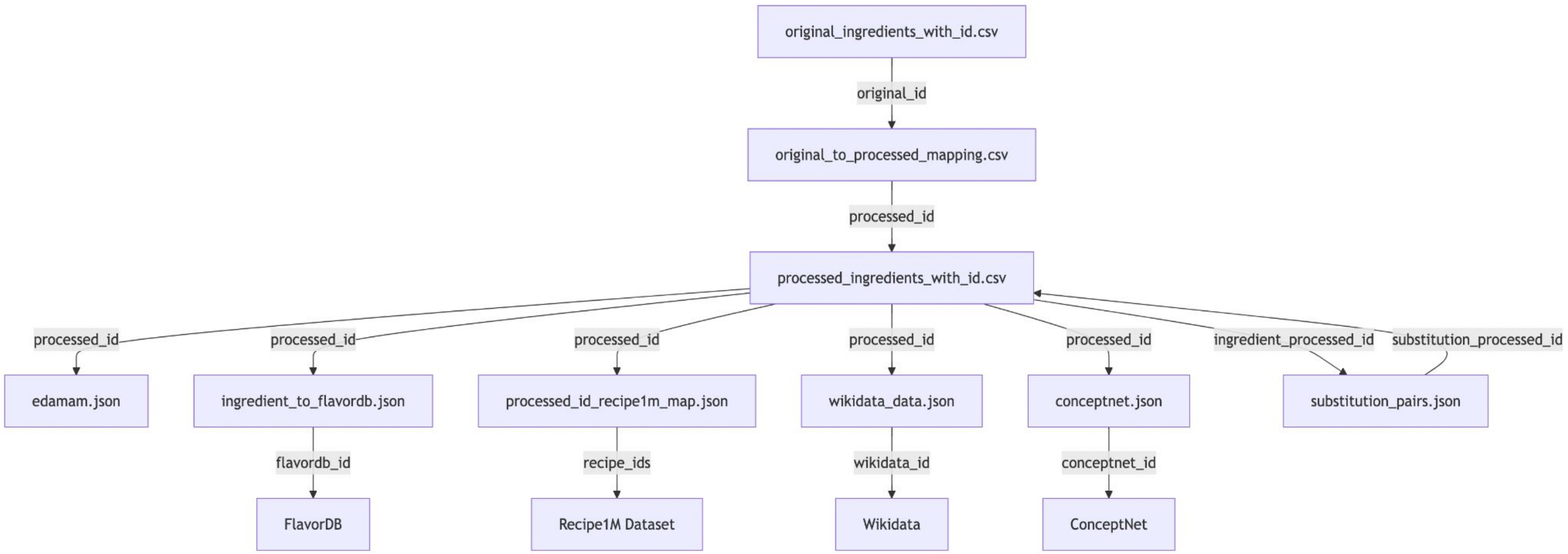
AI-Powered Ingredient Substitute Recommender

Sami & Tim - group 174

Complete Data Relationship Diagram



Dataset 1

Used to find the substitution after model prediction

archive > ds_1.csv > data

```
processed_id,processed,processed_ENERC_KCAL,processed_PROCNT,processed_FAT,processed_CHOCDF,processed_FIBTG,processed_category,processed_label,processed_weight,p
d5a8268e,ababai,-1.0,-1.0,-1.0,-1.0,-1.0,-1,-1,-1.0,-1,-1.0
a5bd8077,abalone,105.0,17.1,0.76,6.01,0.0,Generic foods,Serving,85.0,38,1.0
a8310f99,abalone sauce,-1.0,-1.0,-1.0,-1.0,-1.0,-1,-1,-1.0,38,0.7715134024620056
5d97368d,abalone steak,-1.0,-1.0,-1.0,-1.0,-1.0,-1,-1,-1.0,38,0.7832129001617432
7353da1f,abiu,-1.0,-1.0,-1.0,-1.0,-1.0,-1,-1,-1.0,-1,-1.0
9fb0f62f,abondance,-1.0,-1.0,-1.0,-1.0,-1.0,-1,-1,-1.0,-1,-1.0
8940b6ab,absinthe,231.0,0.0,0.0,0.0,0.0,Generic foods,Serving,50.0,39,0.999999463558197
e5c3d2d4,absolut citron vodka,231.0,0.0,0.0,0.0,0.0,Generic foods,Serving,50.0,40,0.9291608929634094
8f379912,absolut kurant vodka,-1.0,-1.0,-1.0,-1.0,-1.0,-1,-1,-1.0,40,0.7470061182975769
a66607d3,absolut mandarin vodka,231.0,0.0,0.0,0.0,0.0,Generic foods,Serving,50.0,40,0.7385093569755554
318ff5e9,absolut pear flavored vodka,231.0,0.0,0.0,0.0,0.0,Generic foods,Serving,50.0,4708,0.7543702125549316
2d939705,absolut vodka,231.0,0.0,0.0,0.0,0.0,Generic foods,Serving,50.0,40,0.8422659635543823
b8a56b06,abura age,378.0,8.890000343322754,0.0,84.44000244140625,1.100000023841858,Packaged foods,Serving,45.0,-1,-1.0
ecf8fc66,aburage,-1.0,-1.0,-1.0,-1.0,-1.0,-1,-1,-1.0,-1,-1.0
31c3105e,acai pulp,80.0,2.0,6.0,4.0,3.0,Generic foods,Whole,100.0,-1,-1.0
351ec84f,accent seasoning,307.0,9.59,7.53,65.6,11.3,Generic foods,Serving,4.0,-1,-1.0
ea4dacd0,aceite,321.1594863380178,4.870625801645378,11.125212824701885,48.82743612905897,1.6259301938672803,Generic meals,Whole,71.59605640300624,7115,0.71968597
dacee146,aceite de aguacate,83.0,0.0,5.0,8.329999923706055,3.299999952316284,Packaged foods,Serving,30.0,-1,-1.0
7c9a1e59,aceite de canola,-1.0,-1.0,-1.0,-1.0,-1.0,-1,-1,-1.0,-1,-1.0
7d17db74,aceite de oliva,928.3375247959824,0.0,99.97481036264428,0.0,0.0,Packaged foods,Serving,14.003527437778589,-1,-1.0
ab98643b,aceite de oliva antiadherente en aerosol,-1.0,-1.0,-1.0,-1.0,-1.0,-1,-1,-1.0,-1,-1.0
33bba7a7,aceite en aerosol,-1.0,-1.0,-1.0,-1.0,-1.0,-1,-1,-1.0,-1,-1.0
0d22a232,acelga,-1.0,-1.0,-1.0,-1.0,-1.0,-1,-1,-1.0,-1,-1.0
42d3775c,acerola,-1.0,-1.0,-1.0,-1.0,-1.0,-1,-1,-1.0,7114,0.7332984209060669
7f4a5e7a,acesulfame k,378.0,18.7,1.7,73.0,1.1,Generic foods,Serving,31.0,-1,-1.0
d07f0be4,aceto balsamico,107.0,0.0,0.0,20.0,-1.0,Packaged foods,Serving,14.786764781,-1,-1.0
77374c91,achar,11.0,0.33,0.2,2.26,1.2,Generic foods,Serving,65.0,44,0.7015097141265869
bf4fcfc0,achiote,0.0,0.0,0.0,0.0,-1.0,Packaged foods,Serving,1.7999999523162842,44,0.9999997019767761
8e835d75,achiote oil,-1.0,-1.0,-1.0,-1.0,-1.0,-1,-1,-1.0,45,0.8849557638168335
263503c7,achiote paste,185.0,5.0,2.5,35.0,-1.0,Packaged foods,Serving,20.0,46,0.8851694464683533
84893660,achiote powder,0.0,0.0,0.0,0.0,-1.0,Packaged foods,Serving,1.7999999523162842,47,0.8880389332771301
3aeea886,achiote seed,0.0,0.0,0.0,0.0,-1.0,Packaged foods,Serving,1.7999999523162842,44,0.7537855505943298
b85eba2f,achute water,0.0,0.0,0.0,0.0,0.0,Generic foods,Serving,237.0,-1,-1.0
359a1fee,acid blend,800.0,0.0,90.0,0.0,0.0,Packaged foods,Serving,9.857843188,-1,-1.0
d17fc028,acidophilis milk,-1.0,-1.0,-1.0,-1.0,-1.0,-1,-1,-1.0,-1,-1.0
be32739c,acidulated water,0.0,0.0,0.0,0.0,0.0,Generic foods,Serving,237.0,-1,-1.0
c7ada900,acini di pepe,371.0,13.0,1.51,74.7,3.2,Generic foods,Serving,300.0,49,0.7673333287239075
769481fe,acini di pepe pasta,371.0,13.0,1.51,74.7,3.2,Generic foods,Serving,300.0,49,0.8909724950790405
b3d800c7,ackee,66.0,0.83,0.44,16.5,1.3,Generic foods,Serving,190.0,-1,-1.0
86d71b2f,acorn,-1.0,-1.0,-1.0,-1.0,-1.0,-1,-1,-1.0,50,0.9999999403953552
38b1ae53,acorn flour,387.0,6.15,23.9,40.8,-1.0,Generic foods,Serving,28.35,50,0.7003854513168335
cc86eb7f,acorn meal,387.0,6.15,23.9,40.8,-1.0,Generic foods,Serving,28.35,50,0.816230833530426
019064ef,acorn squash,40.0,0.8,0.1,10.4,1.5,Generic foods,Serving,140.0,51,0.8988012671470642
dccc02fd,active compressed yeast,-1.0,-1.0,-1.0,-1.0,-1.0,-1,-1,-1.0,1487,0.7048083543777466
47684931,active dry,-1.0,-1.0,-1.0,-1.0,-1.0,-1,-1,-1.0,-1,-1.0
0ac2f42b,active dry yeast,325.0,40.4,7.61,41.2,26.9,Generic foods,Serving,1.0,2095,0.9207765460014343
7c31f4d3,active starter,259.0,9.49,2.73,48.1,2.1,Generic foods,Serving,57.0,53,0.8965895175933838
c4d9a80e,adobo de los chipotles,-1.0,-1.0,-1.0,-1.0,-1.0,-1,-1,-1.0,1256,0.7516887187957764
d29fab29,adobo sauce,125.88723216107005,4.133187961268788,4.669603004551377,19.933852190125688,6.7328380122382026,Generic meals,Whole,295.9735271865784,54,0.9228
48a73e58,adobo seasoning,-1.0,-1.0,-1.0,-1.0,-1.0,-1,-1,-1.0,55,0.9249363541603088
```

Dataset 2

Used for model training

archive > ds_2.csv > data

```
processed_ENERC_KCAL,processed_PROCNT,processed_FAT,processed_CHOCDF,processed_FIBTG,processed_flavordb_id,processed_cosine_similarity,substitution_ENERC_KCAL,su
-1.0,-1.0,-1.0,-1.0,-1.0,-1,-1.0,43.0,0.47,0.26,10.8,1.7,4600,1.0000001192092896
-1.0,-1.0,-1.0,-1.0,-1.0,-1,-1.0,60.0,0.82,0.38,15.0,1.6,4052,1.0000003576278689
105.0,17.1,0.76,6.01,0.0,38,1.0,-1.0,-1.0,-1.0,-1.0,-1.0,-1,-1.0
105.0,17.1,0.76,6.01,0.0,38,1.0,86.0,14.7,0.96,3.57,0.0,-1,-1.0
105.0,17.1,0.76,6.01,0.0,38,1.0,-1.0,-1.0,-1.0,-1.0,-1.0,1489,0.7724593877792358
105.0,17.1,0.76,6.01,0.0,38,1.0,157.0,21.9,7.02,0.0,0.0,6511,0.8826214671134949
105.0,17.1,0.76,6.01,0.0,38,1.0,27.0,6.7,0.0,0.0,0.0,641,0.8387778997421265
-1.0,-1.0,-1.0,-1.0,-1.0,38,0.7715134024620056,-1.0,-1.0,-1.0,-1.0,-1.0,-1,-1.0
-1.0,-1.0,-1.0,-1.0,-1.0,38,0.7832129001617432,-1.0,-1.0,-1.0,-1.0,-1.0,-1,-1.0
-1.0,-1.0,-1.0,-1.0,-1.0,-1,-1.0,-1.0,-1.0,-1.0,-1.0,-1.0,-1,-1.0
-1.0,-1.0,-1.0,-1.0,-1.0,-1,-1.0,-1.0,-1.0,-1.0,-1.0,-1.0,3127,1.0000001192092896
-1.0,-1.0,-1.0,-1.0,-1.0,-1,-1.0,-1.0,-1.0,-1.0,-1.0,-1.0,2404,1.0000003576278689
-1.0,-1.0,-1.0,-1.0,-1.0,-1,-1.0,-1.0,-1.0,-1.0,-1.0,-1.0,-1,-1.0
231.0,0.0,0.0,0.0,0.0,39,0.999999463558197,-1.0,-1.0,-1.0,-1.0,-1.0,3251,0.9999998211860656
231.0,0.0,0.0,0.0,0.0,39,0.999999463558197,231.0,0.0,0.0,0.0,0.0,4761,1.0
231.0,0.0,0.0,0.0,0.0,39,0.999999463558197,-1.0,-1.0,-1.0,-1.0,-1.0,154,0.999999701976776
231.0,0.0,0.0,0.0,0.0,40,0.9291608929634094,14.0,0.0,0.04,3.59,0.0,4873,0.8864786028862
-1.0,-1.0,-1.0,-1.0,-1.0,40,0.7470061182975769,-1.0,-1.0,-1.0,-1.0,-1.0,-1,-1.0
231.0,0.0,0.0,0.0,0.0,40,0.7385093569755554,-1.0,-1.0,-1.0,-1.0,-1.0,-1,-1.0
231.0,0.0,0.0,0.0,0.0,4708,0.7543702125549316,-1.0,-1.0,-1.0,-1.0,-1.0,-1,-1.0
231.0,0.0,0.0,0.0,0.0,40,0.8422659635543823,-1.0,-1.0,-1.0,-1.0,-1.0,-1,-1.0
378.0,8.890000343322754,0.0,84.44000244140625,1.100000023841858,-1,-1.0,78.0,9.04,4.17,2.85,0.9,2612,0.8290473222732544
-1.0,-1.0,-1.0,-1.0,-1.0,-1,-1.0,-1.0,-1.0,-1.0,-1.0,-1.0,-1,-1.0
80.0,2.0,6.0,4.0,3.0,-1,-1.0,-1.0,-1.0,-1.0,-1.0,-1.0,-1,-1.0
307.0,9.59,7.53,65.6,11.3,-1,-1.0,307.0,9.59,7.53,65.6,11.3,5700,0.8735607266426086
307.0,9.59,7.53,65.6,11.3,-1,-1.0,282.0,14.1,12.9,54.0,34.9,3099,0.8069978952407837
307.0,9.59,7.53,65.6,11.3,-1,-1.0,0.0,0.0,0.0,0.0,-1.0,5687,0.9040606021881104
321.1594863380178,4.870625801645378,11.125212824701885,48.82743612905897,1.6259301938672803,7115,0.719685971736908,-1.0,-1.0,-1.0,-1.0,-1.0,-1,-1.0
83.0,0.0,5.0,8.329999923706055,3.299999952316284,-1,-1.0,-1.0,-1.0,-1.0,-1.0,-1.0,-1,-1.0
-1.0,-1.0,-1.0,-1.0,-1.0,-1,-1.0,-1.0,-1.0,-1.0,-1.0,-1.0,-1,-1.0
928.3375247959824,0.0,99.97481036264428,0.0,0.0,-1,-1.0,-1.0,-1.0,-1.0,-1.0,-1.0,-1,-1.0
-1.0,-1.0,-1.0,-1.0,-1.0,-1,-1.0,-1.0,-1.0,-1.0,-1.0,-1.0,-1,-1.0
-1.0,-1.0,-1.0,-1.0,-1.0,-1,-1.0,-1.0,-1.0,-1.0,-1.0,-1.0,-1,-1.0
-1.0,-1.0,-1.0,-1.0,-1.0,-1,-1.0,-1.0,-1.0,-1.0,-1.0,-1.0,-1,-1.0
-1.0,-1.0,-1.0,-1.0,-1.0,7114,0.7332984209060669,63.0,1.06,0.2,16.0,2.1,1111,0.9999998211860656
378.0,18.7,1.7,73.0,1.1,-1,-1.0,-1.0,-1.0,-1.0,-1.0,-1.0,-1,-1.0
378.0,18.7,1.7,73.0,1.1,-1,-1.0,365.0,2.17,0.0,89.1,0.0,-1,-1.0
378.0,18.7,1.7,73.0,1.1,-1,-1.0,-1.0,-1.0,-1.0,-1.0,-1.0,-1,-1.0
378.0,18.7,1.7,73.0,1.1,-1,-1.0,336.0,0.0,0.0,91.2,0.0,-1,-1.0
378.0,18.7,1.7,73.0,1.1,-1,-1.0,-1.0,-1.0,-1.0,-1.0,-1.0,-1,-1.0
107.0,0.0,0.0,20.0,-1.0,-1,-1.0,-1.0,-1.0,-1.0,-1.0,-1.0,-1,-1.0
11.0,0.33,0.2,2.26,1.2,44,0.7015097141265869,-1.0,-1.0,-1.0,-1.0,-1.0,-1,-1.0
0.0,0.0,0.0,0.0,-1.0,44,0.999999701976776,130.0,21.6,4.81,0.12,0.0,4123,1.0
-1.0,-1.0,-1.0,-1.0,-1.0,45,0.8849557638168335,-1.0,-1.0,-1.0,-1.0,-1.0,-1,-1.0
185.0,5.0,2.5,35.0,-1.0,46,0.8851694464683533,-1.0,-1.0,-1.0,-1.0,-1.0,-1,-1.0
0.0,0.0,0.0,0.0,-1.0,47,0.8880389332771301,-1.0,-1.0,-1.0,-1.0,-1.0,-1,-1.0
0.0,0.0,0.0,0.0,-1.0,44,0.7537855505943298,-1.0,-1.0,-1.0,-1.0,-1.0,-1,-1.0
0.0,0.0,0.0,0.0,0.0,-1,-1.0,-1.0,-1.0,-1.0,-1.0,-1.0,-1,-1.0
800.0,0.0,90.0,0.0,0.0,-1,-1.0,-1.0,-1.0,-1.0,-1.0,-1.0,-1,-1.0
-1.0,-1.0,-1.0,-1.0,-1.0,-1,-1.0,15.0,0.59,1.1,1.15,0.58,103,0.8999112844467163
```

Features and targets

Features (7):

processed_ENERC_KCAL,processed_PROCNT,processed_FAT,processed_CHOCDF,processed_FIBTG,processed_flavordb_id,processed_cosine_similarity

Targets (7):

substitution_ENERC_KCAL,substitution_PROCNT,substitution_FAT,substitution_CHOCDF,substitution_FIBTG,substitution_flavordb_id,substitution_cosine_similarity

Chosen models

- Linear Regression (baseline)
- Support Vector Machine (SVM)
- Random Forest Regression (RFR)
- Neural Network (NN)

LR

Training LR

train_LR.py > ...

```python
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
from sklearn.preprocessing import StandardScaler

# Load dataset
df = pd.read_csv("archive/ds_2.csv")

# Scale the features
scaler = StandardScaler()
scaled_arr = scaler.fit_transform(df)
df_scaled = pd.DataFrame(scaled_arr, columns=df.columns)

# Select the feature and target columns
X = df_scaled[['processed_ENERC_KCAL', 'processed_PROCNT', 'processed_FAT', 'processed_CHOCDF',
        'processed_FIBTG', 'processed_flavordb_id', 'processed_cosine_similarity']]
y = df_scaled[['substitution_ENERC_KCAL', 'substitution_PROCNT', 'substitution_FAT', 'substitution_CHOCDF',
        'substitution_FIBTG', 'substitution_flavordb_id', 'substitution_cosine_similarity']]


# Split the data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize the linear regression model
model = LinearRegression()

# Train the model
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Evaluate the model using mean squared error
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse}")


print("Coefficients:")
print(model.coef_)
print("Intercept:")
print(model.intercept_)



'''
Mean Squared Error: 0.8607558734932768
Coefficients:
[[ 0.37381254 -0.07561138  0.11793094 -0.05367961  0.00399168 -0.04434787
    0.09655912]
 [ 0.23662484  0.29234352 -0.25055133 -0.14491602  0.00345179 -0.04788253
```

LR

Training LR with automatic hyperparameter search

train_LR_auto.py > ...

```python
# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Define hyperparameter grid
param_grid = {'alpha': np.logspace(-4, 2, 10)}  # Alpha values from 0.0001 to 100

# Grid search for ridge regression
ridge = Ridge()
ridge_search = GridSearchCV(ridge, param_grid, cv=3, scoring='neg_mean_squared_error', n_jobs=-1, verbose=1)
ridge_search.fit(X_train, y_train)
best_ridge = ridge_search.best_estimator_
print(f"Best ridge alpha: {ridge_search.best_params_}")

# Grid search for lasso regression
lasso = Lasso(max_iter=5000)  # Ensure Lasso converges
lasso_search = GridSearchCV(lasso, param_grid, cv=3, scoring='neg_mean_squared_error', n_jobs=-1, verbose=1)
lasso_search.fit(X_train, y_train)
best_lasso = lasso_search.best_estimator_
print(f"Best lasso alpha: {lasso_search.best_params_}")

# Train final ridge and lasso models
best_ridge.fit(X_train, y_train)
best_lasso.fit(X_train, y_train)

# Make predictions
y_pred_ridge = best_ridge.predict(X_test)
y_pred_lasso = best_lasso.predict(X_test)

# Evaluate the models
mse_ridge = mean_squared_error(y_test, y_pred_ridge)
mse_lasso = mean_squared_error(y_test, y_pred_lasso)
print(f"Ridge Mean Squared Error: {mse_ridge}")
print(f"Lasso Mean Squared Error: {mse_lasso}")


'''
Fitting 3 folds for each of 10 candidates, totalling 30 fits
Best ridge alpha: {'alpha': 21.54434690031882}
Fitting 3 folds for each of 10 candidates, totalling 30 fits
Best lasso alpha: {'alpha': 0.0001}
Ridge Mean Squared Error: 0.8607540434816144
Lasso Mean Squared Error: 0.8607551729860069
'''
```

SVM

Training SVM

train_SVM.py > ...

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVR
from sklearn.multioutput import MultiOutputRegressor
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import StandardScaler

# Load dataset
df = pd.read_csv("archive/ds_2.csv")

# Scale the features
scaler = StandardScaler()
scaled_arr = scaler.fit_transform(df)
df_scaled = pd.DataFrame(scaled_arr, columns=df.columns)

# Select the feature and target columns
X = df_scaled[['processed_ENERC_KCAL', 'processed_PROCNT', 'processed_FAT', 'processed_CHOCDF',
        'processed_FIBTG', 'processed_flavordb_id', 'processed_cosine_similarity']]
y = df_scaled[['substitution_ENERC_KCAL', 'substitution_PROCNT', 'substitution_FAT', 'substitution_CHOCDF',
        'substitution_FIBTG', 'substitution_flavordb_id', 'substitution_cosine_similarity']]


# Split the data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize the model
svr = SVR(kernel='rbf', C=1.0, epsilon=0.1)

# Make SVR handle multiple outputs
multi_output_svr = MultiOutputRegressor(svr)

# Train the model
multi_output_svr.fit(X_train, y_train)

# Make predictions
y_pred = multi_output_svr.predict(X_test)

# Evaluate the model using mean squared error
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse}")


'''
Mean Squared Error: 0.8866638501044166
'''
```

SVM

Training SVM with automatic hyperparameter search

train_SVM_auto.py > ...

```python
# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Define the hyperparameter search space
param_dist = {
    'estimator__C': np.logspace(-3, 3, 10),  # Regularization parameter
    'estimator__epsilon': np.logspace(-3, 1, 10),  # Epsilon in the loss function
    'estimator__kernel': ['linear', 'poly', 'rbf', 'sigmoid'],  # Kernel types
    'estimator__gamma': ['scale', 'auto']  # Kernel coefficient
}

# Initialize the SVR model
base_svr = SVR()

# Make SVR handle multiple outputs
multi_output_svr = MultiOutputRegressor(base_svr)

# Search for best hyperparameters
random_search = RandomizedSearchCV(
    multi_output_svr,
    param_distributions=param_dist,
    n_iter=20,
    cv=3,
    verbose=1,
    n_jobs=-1,
    scoring='neg_mean_squared_error',
    random_state=42
)

# Fit the model
random_search.fit(X_train, y_train)

# Best parameters
print(f"Best hyperparameters: {random_search.best_params_}")

# Train the best model
best_svr = random_search.best_estimator_
best_svr.fit(X_train, y_train)

# Make predictions
y_pred = best_svr.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse}")
```

RFR

Training RFR

train_RFR.py > ...

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import StandardScaler

# Load dataset
df = pd.read_csv("archive/ds_2.csv")

# Scale the features
scaler = StandardScaler()
scaled_arr = scaler.fit_transform(df)
df_scaled = pd.DataFrame(scaled_arr, columns=df.columns)

# Select the feature and target columns
X = df_scaled[['processed_ENERC_KCAL', 'processed_PROCNT', 'processed_FAT', 'processed_CHOCDF',
        'processed_FIBTG', 'processed_flavordb_id', 'processed_cosine_similarity']]
y = df_scaled[['substitution_ENERC_KCAL', 'substitution_PROCNT', 'substitution_FAT', 'substitution_CHOCDF',
        'substitution_FIBTG', 'substitution_flavordb_id', 'substitution_cosine_similarity']]


# Split the data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

model = RandomForestRegressor(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Evaluate the model using mean squared error
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse}")


'''
Mean Squared Error: 0.673619717571143
'''
```

RFR

Training RFR with automatic hyperparameter search

train_RFR_auto.py > ...

```python
# Split the data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Define hyperparameter grid
param_dist = {
    'n_estimators': [50, 100, 200, 300],  # Number of trees
    'max_depth': [None, 10, 20, 30],  # Depth of trees
    'min_samples_split': [2, 5, 10],  # Minimum samples to split a node
    'min_samples_leaf': [1, 2, 4],  # Minimum samples in leaf node
    'max_features': ['auto', 'sqrt'],  # Number of features per split
    'bootstrap': [True, False]  # Bootstrapping
}

# Search for the best parameters
random_search = RandomizedSearchCV(
    RandomForestRegressor(random_state=42),
    param_distributions=param_dist,
    n_iter=20,
    cv=3,
    verbose=1,
    n_jobs=-1,
    scoring='neg_mean_squared_error'
)

# Fit the model
random_search.fit(X_train, y_train)

# Best hyperparameters
print(f"Best hyperparameters: {random_search.best_params_}")

# Train final model
best_rfr = RandomForestRegressor(**random_search.best_params_, random_state=42)
best_rfr.fit(X_train, y_train)

# Make predictions
y_pred = best_rfr.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
print(f"Mean Squared Error: {mse}")
print(f"Mean Absolute Error: {mae}")


'''
Best hyperparameters: {'n_estimators': 200, 'min_samples_split': 2, 'min_samples_leaf': 4, 'max_features': 'sqrt', 'max_depth': None, 'bootstrap': False}
Mean Squared Error: 0.6697726151041151
Mean Absolute Error: 0.5246457865163581
'''
```

NN

Training NN

train_NN.py > ...

```python
import pandas as pd
from sklearn.model_selection import train_test_split
import tensorflow as tf
from tensorflow import keras
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import StandardScaler

# Load dataset
df = pd.read_csv("archive/ds_2.csv")

# Scale the features
scaler = StandardScaler()
scaled_arr = scaler.fit_transform(df)
df_scaled = pd.DataFrame(scaled_arr, columns=df.columns)

# Select the feature and target columns
X = df_scaled[['processed_ENERC_KCAL', 'processed_PROCNT', 'processed_FAT', 'processed_CHOCDF',
        'processed_FIBTG', 'processed_flavordb_id', 'processed_cosine_similarity']]
y = df_scaled[['substitution_ENERC_KCAL', 'substitution_PROCNT', 'substitution_FAT', 'substitution_CHOCDF',
        'substitution_FIBTG', 'substitution_flavordb_id', 'substitution_cosine_similarity']]


# Split the data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

model = keras.Sequential([
    keras.layers.Dense(64, activation='relu', input_shape=(X.shape[1],)),
    keras.layers.Dense(32, activation='relu'),
    keras.layers.Dense(y.shape[1])
])

# Compile the model
model.compile(optimizer='adam', loss='mse', metrics=['mse'])

# Train the model
model.fit(X_train, y_train, epochs=100, batch_size=8, validation_data=(X_test, y_test), verbose=1)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Evaluate the model using mean squared error (MSE)
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse}")


'''
Mean Squared Error: 0.7408530116081238
'''
```

NN

Training NN with automatic hyperparameter search

train_NN_auto.py > ...

```python
# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Model building function for keras tuner
def build_model(hp):
    model = keras.Sequential()
    model.add(keras.layers.Input(shape=(X.shape[1],)))

    # Number of hidden layers (1-3) and neurons per layer (16-128)
    for i in range(hp.Int('num_layers', 1, 3)):
        model.add(keras.layers.Dense(
            units=hp.Int(f'units_{i}', min_value=16, max_value=128, step=16),
            activation=hp.Choice('activation', ['relu', 'tanh', 'selu'])
        ))

    model.add(keras.layers.Dense(y.shape[1]))

    # Learning rate
    lr = hp.Choice('learning_rate', [0.001, 0.0005, 0.0001])
    optimizer = keras.optimizers.Adam(learning_rate=lr)

    model.compile(optimizer=optimizer, loss='mse', metrics=['mse'])
    return model

# Initialize keras tuner
tuner = kt.Hyperband(
    build_model,
    objective='val_mse',
    max_epochs=50,
    factor=3,
    directory='tuner_results',
    project_name='nn_regression_tuning'
)

# Search for the best hyperparameters
tuner.search(X_train, y_train, validation_data=(X_test, y_test), epochs=50, batch_size=8, verbose=1)

# Get the best model
best_hps = tuner.get_best_hyperparameters(num_trials=1)[0]
best_model = tuner.hypermodel.build(best_hps)

# Train the best model
best_model.fit(X_train, y_train, epochs=100, batch_size=8, validation_data=(X_test, y_test), verbose=1)

# Predict and evaluate
y_pred = best_model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
print(f"Best Model Mean Squared Error: {mse}")

'''
MSE: 0.7183371782302856
```

Results

Best: RFR
Worst: SVM

Metric: MSE

Model performance (Descending order)

1. **RFR auto:** `0.6697726151041151`
2. **RFR:** `0.673619717571143`
3. **NN auto:** `0.7183371782302856`
4. **NN:** `0.7408530116081238`
5. **LR auto:** `0.8607540434816144`
6. **LR:** `0.8607558734932768`
7. **SVM:** `0.8866638501044166`
8. **SVM auto:** `N/A`

> The dataset has 90283 rows, 14 cols.

Usage

Using the model
to find a substitution
for any ingredient

predict_substitution.py > get_substitution_prediction

```python
def get_substitution_prediction(ingredient_features):

    arr = np.array(ingredient_features)
    row = np.hstack([arr[0], np.zeros(7)])
    df1 = pd.DataFrame([row], columns=columns)
    df2 = scaler.transform(df1)
    df3 = pd.DataFrame(df2, columns=columns)
    scaled_features = df3.iloc[:, :7]

    predicted_substitution = model.predict(scaled_features)

    arr = np.array(predicted_substitution)
    row = np.hstack([np.zeros(7), arr[0]])
    df1 = pd.DataFrame([row], columns=columns)
    df2 = scaler.inverse_transform(df1)
    df3 = pd.DataFrame(df2, columns=columns)
    inverse_scaled_features = df3.iloc[:, 7:]

    substitution_features = inverse_scaled_features.iloc[0].to_numpy()

    return substitution_features

def find_closest_ingredient(substitution_features):
    ingredient_features = df_1[feature_cols].values
    substitution_features = np.array(substitution_features).reshape(1, -1)
    distances = cdist(ingredient_features, substitution_features, metric="euclidean")
    closest_idx = np.argmin(distances)

    closest_id = df_1.iloc[closest_idx]["processed_id"]
    closest_name = df_1.iloc[closest_idx]["processed"]

    return closest_id, closest_name


ingredient_features = get_ingredient_params(ingredient_name)

if ingredient_features is not None:
    print('Ingredient features:')
    print(ingredient_features)

    substitution_features = get_substitution_prediction(ingredient_features)
    print('Substitution features:')
    print(substitution_features)

    substitution_id, substitution_name = find_closest_ingredient([substitution_features])
    print('Substitution pair:')
    print(f'{ingredient_name} - {substitution_name}')
```