Analysis 1: Food Dataset from the U.S. Department of Agriculture

Weslley Matheus

Contents

Linear Regression using the Food Dataset from the USDA

Dataset Information

Dataset Information, Cleaning and Transformation

- Foundation Foods Dataset: fdc.nal.usda.gov/download-datasets
- Only 278 food items were analyzed, according to the dataset documentation, mostly natural and unprocessed foods.
- Food items (rows) with missing nutrient values were removed from the dataset sample.
- Specific columns from the dataset – Food Names (e.g: "Lettuce, leaf, green, raw"), Food Categories (e.g: "Vegetables and Vegetable Products"), Food Nutrients (e.g: "4.066 22.046 0.175 0.156 94.008"), and Nutrient Names ("Carbohydrates (g) Energy (kcal) Nitrogen (g) Fats (g) Water (g)") – were merged by shared id columns, which were renamed to not cause compilation problems. These were csv files that were combined into a single dataframe for human readability.
- The resulting dataframe was saved in the file `food_dataset_reshaped.csv` inside the datasets folder `datasets/food_data`.
- All the numerical nutrient columns were standardized through the transformation pipeline.

Data Cleaning: delete, create and merge files and columns

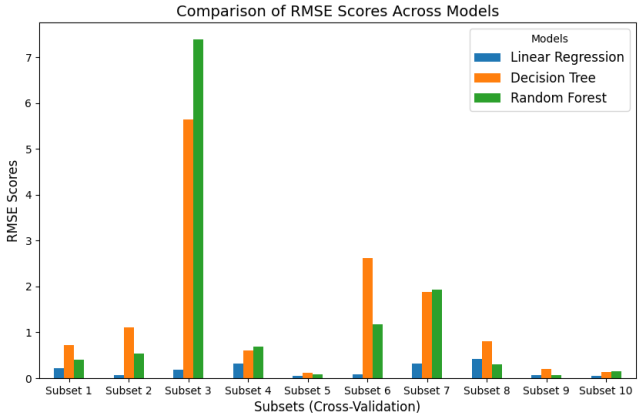
The food data from the USDA came broken down into different files. It was necessary to merge the columns – food names, food categories, nutrient names, nutrient values – by linking them together through shared id columns into a single dataset. This dataset was ordered so the columns – food name, category and nutrient values – were correctly aligned.

Linear Regression of the Protein Column

Root Mean Square Error

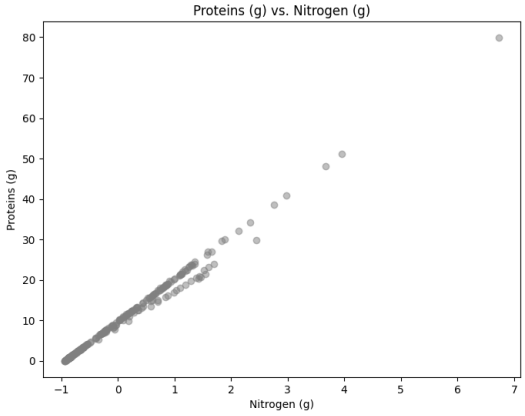
The linear regression of the Protein Column of the Foundation Foods Database from the USDA was done by utilizing only a few major food nutrients – Carbohydrates, Lipids (Fats), Energy, Water, and Nitrogen – as feature columns to predict the values of the label column – Protein.

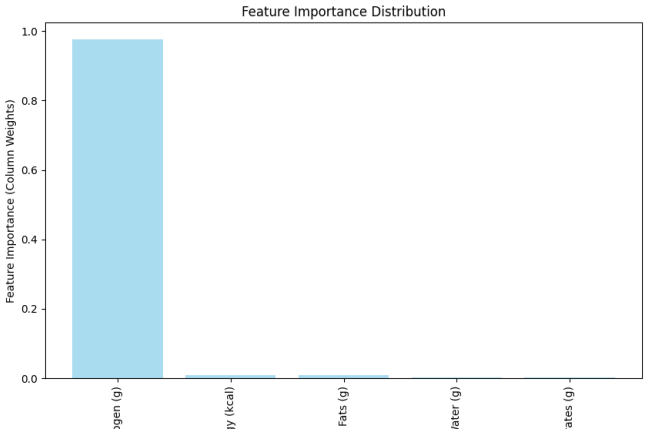
The Linear Regression Algorithm performed better, with lower Root Mean Square Error values across 10 subsets compared to the Decision Tree and Random Forest algorithms. Hence, on these subsets, the Linear Regression Algorithm produced a prediction column with predicted protein values for the food items that were far closer to the original protein values in the label column than those produced by the Decision Tree and Random Forest algorithms.



Linear Correlation and Column Weight

The Nitrogen column had the strongest linear correlation with the protein column and the highest column weight in predicting the values of the protein column.





Note: An attempt was made to utilize every available nutrient value for each food item. However, there were too many missing values for some nutrients (e.g., vitamins, minerals, etc.). Even after performing the imputation of the missing values with the mean of the respective nutrient column, the results were noisy when printing the column weights (feature importances) and the linear correlation between the nutrient columns and the protein column.

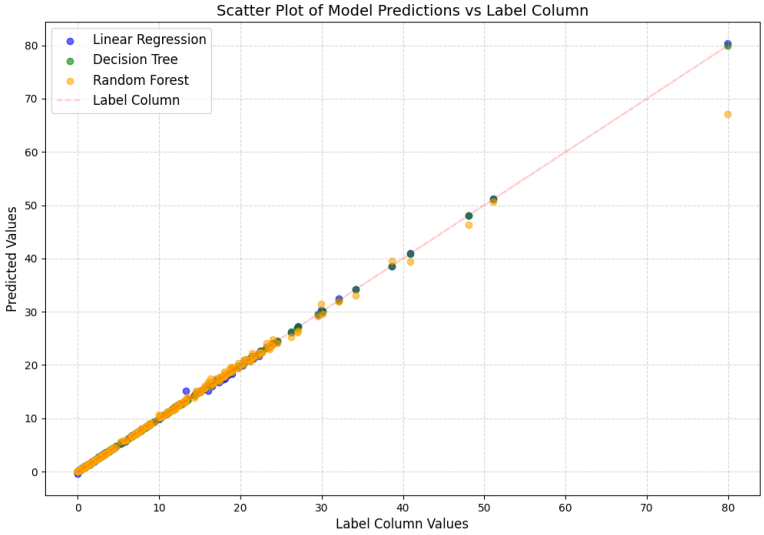
Prediction Columns

The first 10 protein values of the prediction columns made by the 3 algorithms – Linear Regression, Decision Tree, and Random Forest – and the label column with the true values:

```
Linear Regression,Decision Tree,Random Forest,Label Column
9.442165779000208,9.43,9.226184199999985,9.43
12.399209740388855,12.3,12.222536099999987,12.3
5.614927844575728,5.79,5.782374500000002,5.79
17.97849192064918,18.4,18.483971500000024,18.4
```

```
19.955541118286277,20.13125,20.01450529999998,20.13125
17.011649693881935,17.53125,17.6383125,17.53125
17.45081184906512,18.15625,18.216500000000007,18.15625
18.51601668521432,18.74375,18.668838700000027,18.74375
21.600012089636742,21.475,21.47619559999997,21.475
0.6986124293957214,0.65625,0.6533625,0.65625
```

And the next figure shows a scatter plot between the predicted values of the prediction columns made by the algorithms and the original protein values of the label column:



Full Code

Listing 1: Linear Regression of the protein column from the Food Dataset of the USDA.

```
# Python STL
import os
import sys
import tarfile
import urllib.request
import shutil
import typing
import math

# Packages
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Test Datasets
from sklearn.model_selection import train_test_split
from sklearn.model_selection import StratifiedShuffleSplit

# Tranformation Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import LabelBinarizer
from sklearn.preprocessing import StandardScaler
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.pipeline import Pipeline
from sklearn.pipeline import FeatureUnion

# ML Algorithms
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestRegressor

# ML Models
```

```
import joblib
from sklearn.model_selection import GridSearchCV

# Modules




# Dataframe PD Display options
pd.set_option("display.max_columns", None)
pd.set_option("display.max_rows", None)
pd.set_option("display.max_colwidth", None)
pd.set_option("display.width", shutil.get_terminal_size().columns)
# Array NP Display Options
np.set_printoptions(threshold=np.inf)




# Paths
FILE_DIR = os.path.dirname(os.path.abspath(__file__))
PARENT_FILE_DIR = os.path.dirname(FILE_DIR)
PARENT_DIR = os.path.dirname(PARENT_FILE_DIR)
## Datasets
DATA_PATH = os.path.join(PARENT_DIR, "datasets")
FOOD_DATA_PATH = os.path.join(DATA_PATH, "food_data")
## Models
MODEL_DIR = os.path.join(PARENT_DIR, "models")
FOOD_MODEL_DIR = os.path.join(MODEL_DIR, "food_models")
## Images
IMAGES_DIR = os.path.join(PARENT_DIR, "img")
FOOD_IMAGES_DIR = os.path.join(IMAGES_DIR, "food_img")
GRAPHS_IMAGES_DIR = os.path.join(FOOD_IMAGES_DIR, "graphs")

directories = [DATA_PATH, FOOD_DATA_PATH, MODEL_DIR, FOOD_MODEL_DIR, IMAGES_DIR,
FOOD_IMAGES_DIR, GRAPHS_IMAGES_DIR]
for dir in directories:
    os.makedirs(dir, exist_ok=True)




# Download and Load the Dataset
def get_data(data_download: bool = False, data_load: bool = False) -> pd.read_csv:
    """
␣␣␣␣Downloads␣and␣loads␣the␣USDA␣Food␣Data␣Central␣dataset.

␣␣␣␣This␣function␣provides␣two␣functionalities:
␣␣␣␣1.␣Downloads␣and␣extracts␣the␣USDA␣Food␣Data␣Central␣dataset␣from␣the␣official␣URL.
```

```
␣␣␣␣2.␣Loads␣the␣dataset␣into␣pandas␣DataFrames␣if␣the␣files␣have␣already␣been␣
downloaded.

␣␣␣␣Parameters:
␣␣␣␣----------
␣␣␣␣data_download␣:␣bool,␣optional
␣␣␣␣␣␣␣␣If␣set␣to␣True,␣downloads␣the␣dataset␣from␣the␣USDA␣Food␣Data␣Central␣website␣
and␣extracts
␣␣␣␣␣␣␣␣the␣files␣to␣the␣specified␣directory.␣Defaults␣to␣False.

␣␣␣␣data_load␣:␣bool,␣optional
␣␣␣␣␣␣␣␣If␣set␣to␣True,␣loads␣the␣extracted␣dataset␣files␣(e.g.,␣`food.csv`,␣`
food_category.csv`,
␣␣␣␣␣␣␣␣`food_nutrient.csv`,␣and␣`nutrient.csv`)␣into␣pandas␣DataFrames.␣Defaults␣to␣
False.

␣␣␣␣Returns:
␣␣␣␣-------
␣␣␣␣A␣tuple␣of␣DataFrames␣containing␣the␣following␣data:
␣␣␣␣␣␣␣␣-␣`food_csv`:␣Information␣about␣food␣items.
␣␣␣␣␣␣␣␣-␣`food_category_csv`:␣Information␣about␣food␣categories.
␣␣␣␣␣␣␣␣-␣`food_nutrient_csv`:␣Nutritional␣details␣associated␣with␣food␣items.
␣␣␣␣␣␣␣␣-␣`nutrient_csv`:␣Information␣about␣specific␣nutrients.
␣␣␣␣"""

    DATA_URL = "https://fdc.nal.usda.gov/fdc-datasets/
    FoodData_Central_foundation_food_csv_2024-10-31.zip"
    ZIP_FILE_PATH = os.path.join(FOOD_DATA_PATH, "food_data_files.zip")
    EXTRACTED_PATH = os.path.join(FOOD_DATA_PATH, "food_data_files")

    if (data_download):

        urllib.request.urlretrieve(DATA_URL, ZIP_FILE_PATH)
        shutil.unpack_archive(ZIP_FILE_PATH, EXTRACTED_PATH)

        for root, dirs, files in os.walk(EXTRACTED_PATH, topdown=False):

            # Move files to extracted_path
            for file in files:
                src_file_path = os.path.join(root, file)
                dest_file_path = os.path.join(EXTRACTED_PATH, file)
                shutil.move(src_file_path, dest_file_path)

            # Remove empty directories
            if root != EXTRACTED_PATH:
                if not os.listdir(root):
                    os.rmdir(root)

        print(f"\nDataset␣downloaded␣and␣extracted␣to:␣{EXTRACTED_PATH}.\n")
```

```
    if (data_load):

        FOOD_PATH = os.path.join(EXTRACTED_PATH, "food.csv")
        FOOD_CATEGORY_PATH = os.path.join(EXTRACTED_PATH, "food_category.csv")
        FOOD_NUTRIENT_PATH = os.path.join(EXTRACTED_PATH, "food_nutrient.csv")
        NUTRIENT_PATH = os.path.join(EXTRACTED_PATH, "nutrient.csv")

        food_csv = pd.read_csv(FOOD_PATH)
        food_category_csv = pd.read_csv(FOOD_CATEGORY_PATH)
        food_nutrient_csv = pd.read_csv(FOOD_NUTRIENT_PATH, dtype={"footnote": "str"})
        nutrient_csv = pd.read_csv(NUTRIENT_PATH)

        print(f"Dataset␣loaded␣from␣the␣files␣inside:␣{EXTRACTED_PATH}.\n")

        return food_csv, food_category_csv, food_nutrient_csv, nutrient_csv



# Rename Columns
def rename_columns(df: pd.DataFrame, column_suffix_name: str) -> pd.DataFrame:
    """
␣␣␣␣Renames␣the␣columns␣of␣a␣DataFrame␣by␣appending␣a␣given␣suffix␣to␣each␣column␣name.

␣␣␣␣Parameters:
␣␣␣␣␣␣␣␣dataframe␣(pd.DataFrame):␣Input␣DataFrame␣whose␣columns␣need␣renaming.
␣␣␣␣␣␣␣␣column_suffix_name␣(str):␣Suffix␣to␣append␣to␣each␣column␣name.

␣␣␣␣Returns:
␣␣␣␣␣␣␣␣pd.DataFrame:␣DataFrame␣with␣renamed␣columns.

␣␣␣␣Raises:
␣␣␣␣␣␣␣␣TypeError:␣If␣input␣is␣not␣a␣pandas␣DataFrame.
␣␣␣␣"""
    if not isinstance(df, pd.DataFrame):
        raise TypeError("The␣input␣must␣be␣a␣pandas␣DataFrame.")

    return df.rename(columns={col: col + column_suffix_name for col in df.columns})



# Merge DataFrames
def merge_dataframes(df_left: pd.DataFrame, df_right: pd.DataFrame, key_left: str,
key_right: str, how: str = "left") -> pd.DataFrame:
    """
␣␣␣␣Merges␣two␣DataFrames␣on␣specified␣keys␣with␣a␣given␣merge␣method.
```

```
    Parameters:
        df_left (pd.DataFrame): The left DataFrame to be merged.
        df_right (pd.DataFrame): The right DataFrame to be merged.
        key_left (str): The column name in the left DataFrame to use as the merge key.
        key_right (str): The column name in the right DataFrame to use as the merge key
.
        how (str, optional): The type of merge to perform. Default is "left". Other
options are "inner", "outer", and "right".

    Returns:
        pd.DataFrame: The merged DataFrame.

    Raises:
        TypeError: If either input is not a pandas DataFrame.
        ValueError: If the specified keys do not exist in the respective DataFrames.
    """
    if not isinstance(df_left, pd.DataFrame) or not isinstance(df_right, pd.DataFrame):
        raise TypeError("Both inputs must be pandas DataFrames.")

    if key_left not in df_left.columns or key_right not in df_right.columns:
        raise ValueError(f"Specified keys '{key_left}' or '{key_right}' not found in
        the respective DataFrames.")

    return df_left.merge(df_right, left_on=key_left, right_on=key_right, how=how)




# Data Cleaning: Reshape Dataset
def reshape_dataset(food_dataset: pd.DataFrame, save_dataset: bool = False) -> pd.
DataFrame:

    """
    Processes and reshapes a DataFrame containing food nutrient information.

    This function filters the dataset to retain only specific nutrients of interest and
 reshapes
    it into a table where rows correspond to food categories and items, and columns
represent the
    selected nutrient amounts. The resulting dataset is cleaned, formatted, and saved
as a CSV file
    for further analysis. The function returns the reshaped DataFrame.

    Parameters:
    -----------
    food_dataset : pd.DataFrame
        A DataFrame containing detailed information about food items, including their
categories,
        descriptions, nutrient names, and nutrient amounts.
```

```
␣␣␣␣Returns:
␣␣␣␣--------
␣␣␣␣pd.DataFrame
␣␣␣␣␣␣␣␣A␣reshaped␣and␣cleaned␣DataFrame␣where:
␣␣␣␣␣␣␣␣-␣Rows␣represent␣food␣categories␣and␣specific␣food␣items.
␣␣␣␣␣␣␣␣-␣Columns␣represent␣selected␣nutrient␣amounts␣(e.g.,␣proteins,␣carbohydrates,␣
fats,␣energy).
␣␣␣␣␣␣␣␣-␣Rows␣with␣missing␣values␣for␣key␣nutrients␣are␣removed.
␣␣␣␣␣␣␣␣-␣The␣index␣is␣reset␣to␣sequential␣integers␣starting␣from␣0.
␣␣␣␣"""

    # Create the Nurtients Column
    relevant_nutrients = [
        "Protein", "Carbohydrate,␣by␣difference", "Total␣lipid␣(fat)", "Energy␣(Atwater
        ␣General␣Factors)",
        "Water", "Nitrogen"
    ]

    # Get all unique nutrient names dynamically
    all_nutrients = food_dataset["name_nutrient"].unique()

    # Filter the dataset based on all nutrients (noisy) or only on the relevant
    nutrients
    nutrients = food_dataset[food_dataset["name_nutrient"].isin(relevant_nutrients)]

    # Drop columns where all values are missing (Nan)
    nutrients = nutrients.dropna(axis=1, how="all")

    food_dataset_reshaped = nutrients.pivot_table(
        index=["description_food_category", "description_food"],
        columns="name_nutrient",
        values="amount_food_nutrient",
        aggfunc="mean"
    ).reset_index()

    # Rename column names
    food_dataset_reshaped.columns.name = None
    food_dataset_reshaped.rename(
        columns={
            "description_food_category": "Category",
            "description_food": "Food",
            "Protein": "Proteins␣(g)",
            "Carbohydrate,␣by␣difference": "Carbohydrates␣(g)",
            "Total␣lipid␣(fat)": "Fats␣(g)",
            "Energy␣(Atwater␣General␣Factors)": "Energy␣(kcal)",
            "Water": "Water␣(g)",
            "Nitrogen": "Nitrogen␣(g)"
        },
```

```
        inplace=True
    )

    # Remove rows where any relevant nutrient value is Nan
    columns_to_check = list(food_dataset_reshaped.drop(columns={"Category", "Food"}))
    food_dataset_reshaped = food_dataset_reshaped.dropna(subset=columns_to_check)

    # Resets the index of rows to 0, 1, 2, 3, ...
    food_dataset_reshaped.reset_index(drop=True, inplace=True)

    # Convert the dataset to a CSV file
    if (save_dataset):
        food_dataset_reshaped_path = os.path.join(FOOD_DATA_PATH, "
        food_dataset_reshaped.csv")
        food_dataset_reshaped.to_csv(food_dataset_reshaped_path, index=False)
        print(f"\nFile␣converted␣to␣food_dataset_reshaped.csv␣and␣saved␣to␣{
        food_dataset_reshaped_path}.\n")

    return food_dataset_reshaped



# Transformation Pipeline
def transformation_pipeline(columns: list) -> Pipeline:
    """
␣␣␣␣The␣output␣or␣return␣value␣of␣a␣transformer␣becomes␣the␣input␣for␣the␣next␣
transformer.
␣␣␣␣"""

    # Selector: select the nutrients columns
    class CustomDataFrameSelector(BaseEstimator, TransformerMixin):

        def __init__(self, columns):
            self.columns = columns

        def fit(self, dataset, dataset_label = None):
            # self.columns = list(dataset.drop(columns=["Category", "Food", "Proteins (
            g)"]).columns)
            return self

        def transform(self, dataset, dataset_label = None):
            return dataset[self.columns]

    # Tranformation Pipeline
    full_pipeline = Pipeline([
        ("selector", CustomDataFrameSelector(columns)),  # Selects nutrient columns
        ("imputer", SimpleImputer(strategy="mean")),  # Handles missing values
        ("std_scaler", StandardScaler())  # Standardizes the columns
```

```
    ])

    return full_pipeline



# Standardized Columns
def standardization_column(dataset: pd.DataFrame, dataset_numerical_columns: list):
    """
    Check the Standardization of the Columns transformed in the Transformation Pipeline
.
    """
    print("\nStandardization of Columns (Mean = 0, Standard Deviation = 1):")
    for col in list(dataset_numerical_columns):
        print(f"Mean of {col}: ", dataset[col].mean())
        print(f"Standard Deviation of {col}: ", dataset[col].std(), "\n")




# Train Model
def train_models(
    dataset_transformed: np.ndarray,
    dataset_labels: pd.DataFrame,
    save_models: bool
):
    """
    Train, evaluate, and optionally save or load regression models for the dataset.

    Trains the following models:
    1. Linear Regression
    2. Decision Tree
    3. Random Forest

    For each model:
    - Trains on the transformed dataset.
    - Computes RMSE on the training set.
    - Evaluates performance using 10-fold cross-validation and computes RMSE for each
subset.
    - Saves the trained models to disk if `save_models` is True, or loads pre-trained
models if False.

    Parameters:
    ----------
    dataset_transformed : numpy.ndarray
        The preprocessed and transformed feature dataset.
    dataset_labels : numpy.ndarray
        The target labels corresponding to the dataset.
    save_models : bool
```

```
        A flag indicating whether to save the trained models to disk ('True') or load
pre-trained models from disk ('False').
    """

    if (save_models == True):

        # Linear Regression
        lin_reg.fit(dataset_transformed, dataset_labels)
        lin_reg_path = os.path.join(FOOD_MODEL_DIR, "linear_regression_model.pkl")
        joblib.dump(lin_reg, lin_reg_path)
        dataset_predictions_lin = lin_reg.predict(dataset_transformed)
        lin_rmse = np.sqrt(mean_squared_error(dataset_labels, dataset_predictions_lin))
        lin_scores = cross_val_score(lin_reg, dataset_transformed, dataset_labels,
        scoring="neg_mean_squared_error", cv=10)
        lin_scores_rmse = np.sqrt(-lin_scores)

        # Decision Tree
        tree_reg.fit(dataset_transformed, dataset_labels)
        tree_reg_path = os.path.join(FOOD_MODEL_DIR, "decision_tree_model.pkl")
        joblib.dump(tree_reg, tree_reg_path)
        dataset_predictions_tree = tree_reg.predict(dataset_transformed)
        tree_rmse = np.sqrt(mean_squared_error(dataset_labels, dataset_predictions_tree
        ))
        tree_scores = cross_val_score(tree_reg, dataset_transformed, dataset_labels,
        scoring="neg_mean_squared_error", cv=10)
        tree_scores_rmse = np.sqrt(-tree_scores)

        # Random Forest
        forest_reg.fit(dataset_transformed, dataset_labels)
        forest_reg_path = os.path.join(FOOD_MODEL_DIR, "random_forest_model.pkl")
        joblib.dump(forest_reg, forest_reg_path)
        dataset_predictions_forest = forest_reg.predict(dataset_transformed)
        forest_rmse = np.sqrt(mean_squared_error(dataset_labels,
        dataset_predictions_forest))
        forest_scores = cross_val_score(forest_reg, dataset_transformed, dataset_labels
        , scoring="neg_mean_squared_error", cv=10)
        forest_scores_rmse = np.sqrt(-forest_scores)
    else:
        lin_reg_path = os.path.join(FOOD_MODEL_DIR, "linear_regression_model.pkl")
        tree_reg_path = os.path.join(FOOD_MODEL_DIR, "decision_tree_model.pkl")
        forest_reg_path = os.path.join(FOOD_MODEL_DIR, "random_forest_model.pkl")

        # Load pre-trained models
        lin_reg_loaded = joblib.load(lin_reg_path)
        tree_reg_loaded = joblib.load(tree_reg_path)
        forest_reg_loaded = joblib.load(forest_reg_path)

        dataset_predictions_lin = lin_reg_loaded.predict(dataset_transformed)
        dataset_predictions_tree = tree_reg_loaded.predict(dataset_transformed)
```

```
        dataset_predictions_forest = forest_reg_loaded.predict(dataset_transformed)

        lin_rmse = np.sqrt(mean_squared_error(dataset_labels, dataset_predictions_lin))
        lin_scores = cross_val_score(lin_reg_loaded, dataset_transformed,
        dataset_labels, scoring="neg_mean_squared_error", cv=10)
        lin_scores_rmse = np.sqrt(-lin_scores)

        tree_rmse = np.sqrt(mean_squared_error(dataset_labels, dataset_predictions_tree
        ))
        tree_scores = cross_val_score(tree_reg_loaded, dataset_transformed,
        dataset_labels, scoring="neg_mean_squared_error", cv=10)
        tree_scores_rmse = np.sqrt(-tree_scores)

        forest_rmse = np.sqrt(mean_squared_error(dataset_labels,
        dataset_predictions_forest))
        forest_scores = cross_val_score(forest_reg_loaded, dataset_transformed,
        dataset_labels, scoring="neg_mean_squared_error", cv=10)
        forest_scores_rmse = np.sqrt(-forest_scores)

    print("Linear Regression (RMSE of the transformed dataset): ", lin_rmse)
    print("Linear Regression (RMSEs for 10 subsets):")
    print("RMSE for each subset:", lin_scores_rmse)
    print("Mean of the RMSEs:", lin_scores_rmse.mean())
    print("Standard deviation of the RMSEs:", lin_scores_rmse.std())
    print("\n")

    print("Decision Tree (RMSE of the transformed dataset): ", tree_rmse)
    print("Decision Tree (RMSEs for 10 subsets):")
    print("RMSE for each subset:", tree_scores_rmse)
    print("Mean of the RMSEs:", tree_scores_rmse.mean())
    print("Standard deviation of the RMSEs:", tree_scores_rmse.std())
    print("\n")

    print("Random Forest (RMSE of the transformed dataset): ", forest_rmse)
    print("Random Forest (RMSEs for 10 subsets):")
    print("RMSE for each subset:", forest_scores_rmse)
    print("Mean of the RMSEs:", forest_scores_rmse.mean())
    print("Standard deviation of the RMSEs:", forest_scores_rmse.std())
    print("\n")

    column_names = ["Linear Regression", "Decision Tree", "Random Forest"]
    RMSE_columns_arrays = np.column_stack([lin_scores_rmse, tree_scores_rmse,
    forest_scores_rmse])
    RMSE_columns = pd.DataFrame(RMSE_columns_arrays, columns=column_names)

    # Graph of RMSE Scores across subsets
    fig, ax = plt.subplots(figsize=(10, 6))
    RMSE_columns.plot(kind='bar', ax=ax)
    ax.set_title("Comparison of RMSE Scores Across Models", fontsize=14)
```

```
    ax.set_xlabel("Subsets (Cross-Validation)", fontsize=12)
    ax.set_ylabel("RMSE Scores", fontsize=12)
    ax.legend(title="Models", fontsize=12)
    plt.xticks(ticks=range(len(RMSE_columns)), labels=[f"Subset {i+1}" for i in range(
    len(RMSE_columns))], rotation=0)
    plt.savefig(os.path.join(GRAPHS_IMAGES_DIR, "RMSE_scores.png"))



# Prediction Columns
def prediction_columns(dataset_transformed: np.ndarray, dataset_labels: pd.DataFrame):
    """
    Generate and display predictions from multiple regression models (Linear Regression
, Decision Tree,
    and Random Forest) on a given dataset, and print the predictions alongside the true
 labels.

    This function trains three models (Linear Regression, Decision Tree, and Random
Forest) on the provided
    dataset, makes predictions on the same dataset, and prints the first five predicted
 values for each model
    and the true labels for comparison.

    Parameters:
    ----------
    dataset_transformed : np.ndarray
        The transformed dataset, where the features are ready for prediction.

    dataset_labels : pd.DataFrame
        The target labels corresponding to the dataset, containing the true values to
predict.

    Returns:
    ---------
        This function prints the first five predictions from each model and the true
labels to the console.
    """

    lin_reg.fit(dataset_transformed, dataset_labels)
    tree_reg.fit(dataset_transformed, dataset_labels)
    forest_reg.fit(dataset_transformed, dataset_labels)

    dataset_predictions_lin = lin_reg.predict(dataset_transformed)
    dataset_predictions_tree = tree_reg.predict(dataset_transformed)
    dataset_predictions_forest = forest_reg.predict(dataset_transformed)

    column_names = ["Linear Regression", "Decision Tree", "Random Forest", "Label
    Column"]
```

```
    prediction_columns_arrays = np.column_stack([dataset_predictions_lin,
    dataset_predictions_tree, dataset_predictions_forest, dataset_labels])
    prediction_columns = pd.DataFrame(prediction_columns_arrays, columns=column_names)

    print("Prediction Columns:")
    print(prediction_columns.iloc[0:5])

    # Save the prediction columns dataframe as a csv file
    prediction_columns_path = os.path.join(FOOD_DATA_PATH, "prediction_columns.csv")
    prediction_columns.to_csv(prediction_columns_path, index=False)
    print(f"\nPrediction columns converted to prediction_columns.csv and saved to {
    prediction_columns_path}.\n")

    # Scatter plot: prediction columns vs label column
    plt.figure(figsize=(12,8))
    plt.scatter(prediction_columns["Label Column"], prediction_columns["Linear
    Regression"],color="blue",alpha=0.6,label="Linear Regression")
    plt.scatter(prediction_columns["Label Column"], prediction_columns["Decision Tree"
    ],color="green",alpha=0.6,label="Decision Tree")
    plt.scatter(prediction_columns["Label Column"], prediction_columns["Random Forest"
    ],color="orange",alpha=0.6,label="Random Forest")
    plt.plot(prediction_columns["Label Column"],prediction_columns["Label Column"],
    color="red",alpha=0.2,linestyle="--",label="Label Column")
    plt.title("Scatter Plot of Model Predictions vs Label Column", fontsize=14)
    plt.xlabel("Label Column Values", fontsize=12)
    plt.ylabel("Predicted Values", fontsize=12)
    plt.legend(fontsize=12)
    plt.grid(True, linestyle="--", alpha=0.5)
    plt.savefig(os.path.join(GRAPHS_IMAGES_DIR, "scatter_plot_prediction_columns.png"))



# Fine Tune Model
def fine_tune_model(
    dataset_transformed: np.ndarray,
    dataset_labels: pd.DataFrame,
    model: BaseEstimator,
    save_model: bool,
    model_name = "model",
) -> BaseEstimator:
    """
    Fine-tune a given machine learning model by performing hyperparameter tuning with
GridSearchCV.

    This function searches for the best parameters for the model using cross-validation
 and grid search,
    evaluates the model performance on the training data, and calculates the feature
importance. It also tests
```

```
␣␣␣␣the␣model␣on␣a␣separate␣test␣dataset␣and␣saves␣the␣fine-tuned␣model␣if␣requested.

␣␣␣␣Parameters:
␣␣␣␣----------
␣␣␣␣dataset_transformed␣:␣np.ndarray
␣␣␣␣␣␣␣␣The␣transformed␣training␣dataset,␣with␣features␣ready␣for␣model␣training.

␣␣␣␣dataset_labels␣:␣pd.DataFrame
␣␣␣␣␣␣␣␣The␣target␣labels␣corresponding␣to␣the␣dataset,␣containing␣the␣true␣values␣to␣
predict.

␣␣␣␣model␣:␣BaseEstimator
␣␣␣␣␣␣␣␣The␣machine␣learning␣model␣to␣be␣fine-tuned.␣This␣model␣must␣implement␣the␣`fit
()`␣and␣`predict()`␣methods
␣␣␣␣␣␣␣␣from␣scikit-learn's␣`BaseEstimator`.

␣␣␣␣save_model␣:␣bool
␣␣␣␣␣␣␣␣If␣True,␣the␣fine-tuned␣model␣will␣be␣saved␣to␣a␣file.

␣␣␣␣model_name␣:␣str,␣optional␣(default="model")
␣␣␣␣␣␣␣␣The␣base␣name␣to␣be␣used␣for␣saving␣the␣fine-tuned␣model.

␣␣␣␣Returns:
␣␣␣␣-------
␣␣␣␣best_model␣:␣BaseEstimator
␣␣␣␣␣␣␣␣The␣fine-tuned␣model␣after␣grid␣search,␣ready␣for␣predictions.

␣␣␣␣Notes:
␣␣␣␣-----
␣␣␣␣-␣This␣function␣uses␣GridSearchCV␣to␣find␣the␣best␣combination␣of␣hyperparameters␣
for␣the␣given␣model.
␣␣␣␣-␣The␣model's␣feature␣importances␣are␣printed,␣showing␣the␣contribution␣of␣each␣
column␣to␣the␣column␣of␣predictions.
␣␣␣␣-␣If␣`save_model`␣is␣set␣to␣True,␣the␣fine-tuned␣model␣is␣saved␣as␣a␣`.pkl`␣file␣in
␣the␣`models/housing`␣directory.
␣␣␣␣"""

    # Search for the best parameters for the model: minimum RMSE and best performance
    across the subsets
    param_grid = [
        {"n_estimators":[3,10,30], "max_features":[2,4,6,8]},
        {"bootstrap":[False], "n_estimators":[3,10], "max_features":[2,3,4]}
    ]
    grid_search = GridSearchCV(model, param_grid, cv=5, scoring="neg_mean_squared_error
    ")
    grid_search.fit(dataset_transformed, dataset_labels)

    best_parameters = grid_search.best_params_
    best_model = grid_search.best_estimator_
```

```
    best_model_predictions = best_model.predict(dataset_transformed)
    best_model_rmse = np.sqrt(mean_squared_error(dataset_labels, best_model_predictions
    ))

    cv_results = grid_search.cv_results_
    print("Mean␣of␣the␣RMSEs␣of␣the␣subsets␣for␣each␣parameter␣combination:")
    for mean_score, params in zip(cv_results["mean_test_score"], cv_results["params"]):
        print("Mean␣of␣the␣RMSEs:␣", np.sqrt(-mean_score), "for␣parameters:␣", params)

    print("Best␣parameters␣for␣the␣model:␣", best_parameters)
    print("Best␣model␣RMSE:␣", best_model_rmse)

    if (save_model):
        model_full_name = model_name + "_fine_tuned_model" + ".pkl"
        fine_tuned_model_path = os.path.join(FOOD_MODEL_DIR, model_full_name)
        joblib.dump(best_model, fine_tuned_model_path)
        print("\nFine-tuned␣model␣saved␣at:", fine_tuned_model_path, "\n")

    return best_model




# Column Weights
def column_weights(
    model: BaseEstimator,
    dataset_transformed: pd.DataFrame = None
):
    """
␣␣␣␣Calculates␣the␣importance␣(weights)␣of␣each␣column␣based␣on␣the␣trained␣model's␣
feature␣importances.

␣␣␣␣Parameters
␣␣␣␣----------
␣␣␣␣model␣:␣BaseEstimator
␣␣␣␣␣␣␣␣A␣trained␣machine␣learning␣model␣with␣a␣`feature_importances_`␣attribute.
␣␣␣␣dataset_transformed␣:␣pd.DataFrame,␣optional
␣␣␣␣␣␣␣␣The␣processed␣dataset␣(numerical␣and␣textual␣features)␣transformed␣in␣the␣
transformation␣pipeline␣and␣used␣by␣the␣ML␣algorithms.

␣␣␣␣Returns
␣␣␣␣-------
␣␣␣␣None
␣␣␣␣␣␣␣␣Prints␣the␣feature␣importance␣weights␣and␣their␣corresponding␣columns,␣sorted␣
in␣descending␣order␣of␣importance.

␣␣␣␣Notes
␣␣␣␣-----
```

```
    - Ensure the model is already trained and includes a `feature_importances_` attribute.
    - `dataset_processed` should match the features used to train the model.
    """
    # Extract feature importance weights from the model
    column_weights = [float(weight) for weight in model.feature_importances_]

    # Get the column names of the transformed dataset
    columns = list(dataset_transformed.columns)

    # Pair and sort columns with their corresponding weights in descending order
    sorted_column_weights = sorted(zip(column_weights, columns), reverse=True)

    print("Feature Importances (Column Weights):")
    print(sorted_column_weights, "\n")

    # Feature Importance Graph
    # Extract feature names and weights
    weights, features = zip(*sorted_column_weights)
    plt.figure(figsize=(10, 6))
    plt.bar(features, weights, color='skyblue', alpha=0.7)
    plt.xticks(rotation=90)  # Rotate feature names for better readability
    plt.xlabel('Features (Column Names)')
    plt.ylabel('Feature Importance (Column Weights)')
    plt.title('Feature Importance Distribution')
    plt.savefig(os.path.join(GRAPHS_IMAGES_DIR, "column_weights.png"))



# Linear Correlation
def linear_correlation(dataset_numerical: pd.DataFrame, column_name: str, save_graphs:
bool):

    # Compute the correlation matrix
    corr_matrix = dataset_numerical.corr()
    linear_corr = corr_matrix[column_name].sort_values(ascending=False)

    # Plot scatter plots for each feature
    for col in dataset_numerical:
        if col == column_name:
            continue
        plt.figure(figsize=(8, 6))
        plt.scatter(dataset_numerical[col], dataset_numerical[column_name], alpha=0.5,
        c="gray")
        plt.title(f"{column_name} vs. {col}")
        plt.xlabel(col)
        plt.ylabel(column_name)
```

```
        plt.savefig(os.path.join(GRAPHS_IMAGES_DIR, f"linear_correlation_{column_name}_
        {col}.png"))

    return linear_corr



if __name__ == "__main__":

    # sys.exit("Sys.exit(): stops the code right here.")

    # Download and load the dataset files as DataFrames
    food_csv, food_category_csv, food_nutrient_csv, nutrient_csv = get_data(
    data_download=True, data_load=True)

    # Add suffixes to column names to indicate their source
    food_csv = rename_columns(food_csv, "_food")
    food_category_csv = rename_columns(food_category_csv, "_food_category")
    food_nutrient_csv = rename_columns(food_nutrient_csv, "_food_nutrient")
    nutrient_csv = rename_columns(nutrient_csv, "_nutrient")

    # Merge DataFrames to combine food, categories, and nutrient information
    food_category_df = merge_dataframes(food_category_csv, food_csv, "id_food_category"
    , "food_category_id_food")
    food_nutrient_df = merge_dataframes(food_nutrient_csv, nutrient_csv, "
    nutrient_id_food_nutrient", "id_nutrient")
    food_merged_df = merge_dataframes(food_category_df, food_nutrient_df, "fdc_id_food"
    , "fdc_id_food_nutrient")

    print("Food␣Dataset␣merged␣dataframes:")
    print(food_merged_df.iloc[0:1], "\n")

    # Reshape the merged dataset: create 'Category', 'Food', and 'Nutrients' columns,
    and remove unnecessary columns
    food_dataset_reshaped = reshape_dataset(food_dataset=food_merged_df, save_dataset=
    True)
    print(f"Food␣Dataset␣reshaped␣{food_dataset_reshaped.shape}:")
    print(food_dataset_reshaped.iloc[0:1], "\n")

    # Split the dataset into label, nutrient, and textual columns
    food_dataset = food_dataset_reshaped.drop(columns=["Proteins␣(g)"])
    food_dataset_labels = food_dataset_reshaped["Proteins␣(g)"].copy()
    food_dataset_nutrients = food_dataset.drop(columns=["Category", "Food"])
    food_dataset_text = food_dataset.drop(columns=food_dataset_nutrients.columns)

    print("Nutrient␣columns␣to␣be␣included␣in␣the␣transformation␣pipeline:")
    print(list(food_dataset_nutrients.columns), "\n")
```

```
# Transformation Pipeline
full_pipeline = transformation_pipeline(list(food_dataset_nutrients.columns))
full_pipeline.fit(food_dataset_reshaped)
food_dataset_nutrients_transformed = full_pipeline.transform(food_dataset_reshaped)

# Merge the text, transformed nutrient columns, and labels into a single dataFrame
food_dataset_nutrients_transformed_df = pd.DataFrame(
food_dataset_nutrients_transformed, columns=food_dataset_nutrients.columns)
food_dataset_transformed = pd.concat([food_dataset_text,
food_dataset_nutrients_transformed_df, food_dataset_labels], axis=1)

print("Food␣Dataset␣after␣the␣transformation␣pipeline:")
print(food_dataset_transformed.iloc[0:1], "\n")

# Check the standardization of the columns
print("Standardization␣of␣the␣nutrient␣columns␣made␣by␣the␣transformation␣pipeline
:\n")
standardization_column(food_dataset_transformed, list(food_dataset_nutrients.
columns))

# Train the models: use the ML algorithms to create the prediction column

# ML Algorithms
lin_reg = LinearRegression()
tree_reg = DecisionTreeRegressor()
forest_reg = RandomForestRegressor()

print("Root␣Mean␣Square␣Errors␣of␣the␣ML␣Algorithms:\n")
train_models(
    dataset_transformed = food_dataset_nutrients_transformed,
    dataset_labels = food_dataset_labels,
    save_models = True
)

print("Prediction␣Columns␣created␣by␣the␣ML␣Algorithms:\n")
prediction_columns(
        dataset_transformed = food_dataset_nutrients_transformed,
        dataset_labels = food_dataset_labels
    )

print("Results␣of␣the␣best␣model␣(Random␣Forest␣Algorithm):\n")
best_model = fine_tune_model(
    dataset_transformed = food_dataset_nutrients_transformed,
    dataset_labels = food_dataset_labels,
    model = forest_reg,
    save_model = True,
    model_name = "random_forest"
)
```

```
print("Column Weights (Feature Importances) created by the best model (Random
Forest Algorithm):")
column_weights(
    model = best_model,
    dataset_transformed = food_dataset_transformed.drop(columns=["Category", "Food"
    , "Proteins (g)"])
)

print("Linear Correlation between the processed nutrient columns and the label
column (Proteins):")
linear_corr_protein = linear_correlation(
    dataset_numerical=food_dataset_transformed.drop(columns=["Category", "Food"]),
    column_name="Proteins (g)",
    save_graphs=True
)
print(linear_corr_protein)
```

Appendix

Formulas

Mean

$$m = \frac{1}{n}\sum_{i=1}^{n} v_i$$

Variance

$$V = \frac{1}{n}\sum_{i=1}^{n}(v_i - m)^2$$

- Deviation from the mean: $v_i - mean$

Standard Deviation

$$SD = \sqrt{V}$$

Root Mean Square Error

$$RMSE(\boldsymbol{X}, h) = \sqrt{\frac{1}{m}\sum_{i=1}^{m}\left(h(x^{(i)}) - y^{(i)}\right)^2}$$

$$RMSE(\boldsymbol{Dataset}, MLAlgorithm) = \sqrt{\frac{1}{rows}\sum_{i=1}^{rows}\left(MLAlgorithm(predicted\,value^{(i)}) - label\,value^{(i)}\right)^2}$$

- Euclidean distance: straight line $d = \sqrt{\Delta x^2 + \Delta y^2}$
- The ML Algorithm takes into consideration all the column values of the dataset to form a column of predicted values.
- The RMSE measures the standard deviation of the predicted values from the label values.

Mean Absolute Error

$$MAE(\boldsymbol{X}, h) = \frac{1}{m} \sum_{i=1}^{m} \mid h(x^{(i)}) - y^{(i)} \mid$$

$$MAE(Dataset, MLAlgorithm) = \frac{1}{rows} \sum_{i=1}^{rows} \mid MLAlgorithm(predictedvalue^{(i)}) - labelvalue^{(i)} \mid$$

- Manhattan distance: grid $d = \mid \Delta x \mid + \mid \Delta y \mid$
- Both the RMSE and the MAE are ways to measure the distance between two vectors: the column of predicted values from the column of label values.
- The mean absolute error is preferred when the data has many outliers.

Difference between RMSE and Standard Deviation:

$$RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^{n} \left(y_{predicted}^{(i)} - y_{label}^{(i)} \right)^2}$$

$$STD = \sqrt{\frac{1}{n} \sum_{i=1}^{n} \left(y_{predicted}^{(i)} - mean \right)^2}$$

RMSE (Root Mean Squared Error) measures the average magnitude (value) of the differences (errors) between the predicted values and the true values (labels). In other words, it's the average "distance" between the predicted values and the label values. It is the deviation from the label.

Standard Deviation measures the average distance of the differences between the predicted values from their own mean. It measures how spread out the values (in a dataset) are from the mean value. When applied to predictions, it measures how spread out the predicted values are from their own mean. It is the deviation from the mean.

Standardization of a Column

$$Column = V_0, V_1, V_2, V_3, ...V_n \rightarrow Column' = Z_0, Z_1, Z_2, Z_3, ...Z_n$$

$$Z_i = \frac{V_i - mean(Column)}{Standard\ Deviation(Column)}$$

$$Mean(Column') = \frac{1}{n}\sum_{i=1}^{n}(Z_i) \simeq 0$$

$$SD(Column') = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(Z_i - mean')^2} \simeq 1$$