<https://stackoverflow.com/questions/70200476/github-actions-every-step-must-define-a-uses-or-run-key>

For formatting the yaml file instructions.

I removed the redundant columns from the variable columns

"""

drop\_list = drop\_columns + useless\_column *# initialises the list to drop the items from the dataframe*

*# initialise the practise columns.*

practice\_list = ["legum","conc","add","lact","breed","covman","comp","drag","cov","plow","solar","biog","ecodr"]

for feat in first\_line: *# iterate through the first line columns*

    if **any**(x in feat for x in practice\_list): *# checks if the features are in the practice\_list*

        drop\_list.**append**(feat) *# adds the features onto the drop-list*

*# print("Feature: {}".format(feat)) # test displays the features that will be added into the useless column*

*# compare drop\_list with columns*

**print**("Columns: {}".**format**(**len**(columns)))

**print**("Drop\_list: {}".**format**(**len**(drop\_list)))

*# removes items that start with "net\_name" from the variable columns*

columns = [feat for feat in columns if "net\_name" not in feat]

for feat in drop\_list: *# iterate through the drop\_list*

    if feat in columns:

**print**("Feature: {}".**format**(feat))

*# df = df.drop(feat) # drop the feature from the dataframe*

Created a function to process the data headings

GOAL: encapsulate the heading processes into a function. – COMPLETE

I used the function collect\_headings to process the data frame to extract the necessary information for grouping the data into different columns:

def **collect\_headings**(df):

    """

    Function uses the dataframe to collect the headings from the csv file

    @param: df - dataframe

    @return: headings - list of headings

    """

*# get the first line from farmers.csv*

    first\_line = df.iloc[0]

    columns\_test = first\_line.to\_string().split(',') *# split the first line into a list by comma*

*# print the first element of the list with the first character removed*

    column\_headings = columns\_test[0].split('\n')

    real\_headings = [] *# intialises a list to store the heading variables*

    for heading in column\_headings: *# iterates through the unprocessed column headings*

*# split the heading by any whitespace*

        heading = heading.split()

*# append the second element of the list to the real\_headings list*

        real\_headings.**append**(heading[1])

    return real\_headings

Implement a function to process the data in the dataset before using pandas to format the data structures.

I implemented a function to convert the loaded data frame into suitable data formats by iterating through the datasets and eliminating the newline and the spaces functions.

Graphical user interface, text, application

Description automatically generated

<https://realpython.com/python-csv/>

Researched this site to help with improving how to process the data.

Graphical user interface, text, application

Description automatically generated

I generated the column names by reading the files with the csv function.

Graphical user interface, application, table

Description automatically generated

I wrote a test that displayed the data in table format through the code:

def **read\_data**():

    """

    Reads the data from the csv file

    @param: None

    @return: dataframe with the data that has been loaded into the csv file "data.csv"

    """

    file\_name = 'farmers.csv' *# name of the csv file*

    with **open**(file\_name) as csv\_file: *# open the csv file*

        csv\_reader = **csv**.**reader**(csv\_file, delimiter = ',') *# read the csv file*

*# intialise a variable to track the number of lines in the csv file*

        line\_count = 0

        for row in csv\_reader: *# iterate through the csv file*

            if line\_count == 0:

**print**(f'Column names are {", ".**join**(row)}') *# displays the column names*

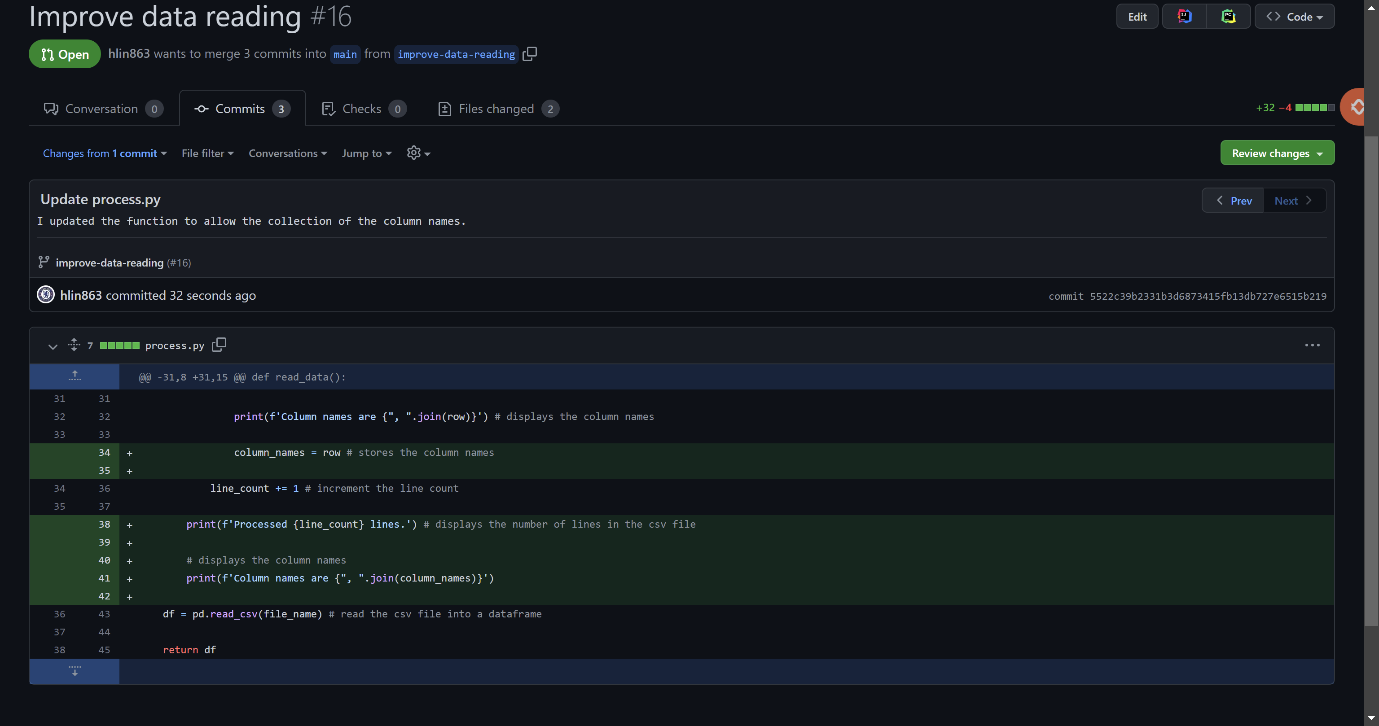
            line\_count += 1 *# increment the line count*

    df = **pd**.**read\_csv**(file\_name) *# read the csv file into a dataframe*

    return df

df = **read\_data**()

**print**(df)



Used data frame to process the individual columns.

I generated the function to clean up the datasets for later operations.

I created a function to apply logistic regression on the cleaned datasets after removing the null values and splitting the data into training and testing sets.

*# import the Linear Regression class*

from **sklearn**.**linear\_model** import **LinearRegression**

*# import the sklearn library to preprocess the data*

from **sklearn** import **preprocessing**

*# import the pandas library*

import **pandas** as **pd**

*# import the train\_test\_split function*

from **sklearn**.**model\_selection** import **train\_test\_split**

*# import the matplotlib library*

import **matplotlib**.**pyplot** as **plt**

*# load the farmers\_cleaned.csv dataset*

df = **pd**.**read\_csv**('farmers\_cleaned.csv')

*# create a Linear Regression model*

model = **LinearRegression**()

df.**dropna**(inplace=True)

*# intialise the input and output variables: X and y*

X = df.**to\_numpy**() *# converts the dataframe to a numpy array*

X = **preprocessing**.**scale**(X) *# preprocess the data*

y = df.**pop**('cons\_general').**to\_numpy**() *# removes the net\_name column from the dataframe and converts it to a numpy array*

X\_train, X\_test, y\_train, y\_test = **train\_test\_split**(X, y, test\_size=0.2, random\_state=0)

*# fit the model to the data*

model.**fit**(X\_train, y\_train)

*# print the model's score*

**print**("The score for the Linear Regression model is: ", model.**score**(X\_test, y\_test))

*# visualise the model's predictions*

**plt**.**scatter**(y\_test, model.**predict**(X\_test), color='black')

**plt**.**xlabel**('Actual Consumption') *# set the x-axis label*

**plt**.**ylabel**('Predicted Consumption') *# set the y-axis label*

**plt**.**title**('Actual Consumption vs Predicted Consumption') *# set the title*

**plt**.**show**()