

CS2 - Regional Health and Nutrition Case Study

DS 4002 - Fall 2023 - Shreya Nagabhirava

Due: December 11, for comments; final December 17 Submission format: Link to Github repository (submit to Canvas under the posted assignment) and presentation in class

Individual Assignment

General Description: Submit to canvas a link to your case study repository and a present your slideshow in class

Preparatory Assignments: Everything in the course, but especially the case study review.

Why am I doing this? This case study will help you become more familiar with the life cycle of data science research and analysis. It will help you to answer a research question. This case study focuses on using a time-series data set to complete analysis using code. It also requires you to create an appealing and interesting presentation for the audience that is easily understandable. Read through the materials given to you to help find a solution to the research question at hand!

- Course Learning Objective: Analyze a time-series data set
- Course Learning Objective: Practice coding to manipulate data
- Course Learning Objective: Create a machine learning model
- Course Learning Objective: Practice public speaking and create a comprehensive presentation

What am I going to do? Read the one-page prompt you are given for this case study. That will give you an idea of what the question you are trying to answer is and the resources you have to answer it. Then, you have to look through the articles and extra resources provided to you. While doing this, take note of what you might want to use to create your deliverable. Ask questions and make a plan for your analysis. Once you complete the analysis, create your presentation and all your materials to your github repository so that you will be prepared for your final presentation!

Tips for success:

- Ask lots of questions! Don't be afraid to ask how other people are approaching the problem because you may learn things you didn't know before.
- Talk to your professor and TA. They will make sure you are on the right track.

How will I know I have Succeeded? You will meet expectations on this assignment when you follow the criteria in the rubric below.

Spec Category	Spec Details
Formatting	<ul style="list-style-type: none"> ● Repository - Create a github repository containing all the materials organized how we described below <ul style="list-style-type: none"> ○ Submit a link to the repo ○ Contents: <ul style="list-style-type: none"> ■ README.md ■ SRC folder ■ DATA folder ■ Presentation ■ LICENSE.md ○ Use pdf format when possible ○ For code and data products use the most applicable, appropriate format
README.md	<ul style="list-style-type: none"> ● Goal: The README file gives context to the case study, the steps taken during analysis, data source used, and any images or deliverables that you created. <ul style="list-style-type: none"> ○ Explain any data manipulation or pre-processing that you completed ○ Include visuals or your model from your code <ul style="list-style-type: none"> ■ Accuracy of the training set and testing set ○ Highlight important findings ○ Cite resources used
SRC	<ul style="list-style-type: none"> ● Goal: This is a folder in your github repo that stores all of your code <ul style="list-style-type: none"> ○ Code does not have to be clean, but other students should be able to read and understand the code so they can replicate this analysis
DATA	<ul style="list-style-type: none"> ● Goal: This is a folder in your github repo that stores the data source you used for this assignment <ul style="list-style-type: none"> ○ This is given to you already, just make sure to have a copy in your project repo
Presentation	<ul style="list-style-type: none"> ● Goal: This is a PowerPoint presentation that you will present to your class. It should be an overview of the work you completed and your findings. <ul style="list-style-type: none"> ○ Include any struggles you faced, your process, and final outcome
LICENSE.md	<ul style="list-style-type: none"> ● Goal: This file explains to visitors of your repository what they are allowed to use and how they should cite your repository. (MIT License)

Python Packages for Linear Regression

It's time to start implementing linear regression in Python. To do this, you'll apply the proper packages and their functions and classes.

NumPy is a fundamental Python scientific package that allows many high-performance operations on single-dimensional and multidimensional arrays. It also offers many mathematical routines. Of course, it's open-source.

If you're not familiar with NumPy, you can use the official [NumPy User Guide](#) and read [NumPy Tutorial: Your First Steps Into Data Science in Python](#). In addition, [Look Ma, No for Loops: Array Programming With NumPy](#) and [Pure Python vs NumPy vs TensorFlow Performance Comparison](#) can give you a good idea of the performance gains that you can achieve when applying NumPy.

The package **scikit-learn** is a widely used Python library for machine learning, built on top of NumPy and some other packages. It provides the means for preprocessing data, reducing dimensionality, implementing regression, classifying, clustering, and more. Like NumPy, scikit-learn is also open-source.

You can check the page [Generalized Linear Models](#) on the [scikit-learn website](#) to learn more about linear models and get deeper insight into how this package works.

If you want to implement linear regression and need functionality beyond the scope of scikit-learn, you should consider **statsmodels**. It's a powerful Python package for the estimation of statistical models, performing tests, and more. It's open-source as well.

You can find more information on statsmodels on [its official website](#).

Now, to follow along with this tutorial, you should install all these packages into a [virtual environment](#):

Shell




```
(venv) $ python -m pip install numpy scikit-learn statsmo
```

This will install NumPy, scikit-learn, statsmodels, and their dependencies.

Step 1: Import packages and classes

The first step is to import the package `numpy` and the class `LinearRegression` from `sklearn.linear_model`:

```
Python   
  
>>> import numpy as np  
>>> from sklearn.linear_model import LinearRegression
```


Now, you have all the functionalities that you need to implement linear regression.

The fundamental data type of NumPy is the array type called `numpy.ndarray`. The rest of this tutorial uses the term **array** to refer to instances of the type `numpy.ndarray`.

You'll use the class `sklearn.linear_model.LinearRegression` to perform linear and polynomial regression and make predictions accordingly.

Step 2: Provide data

The second step is defining data to work with. The inputs (regressors, x) and output (response, y) should be arrays or similar objects. This is the simplest way of providing data for regression:

```
Python   
  
>>> x = np.array([5, 15, 25, 35, 45, 55]).reshape((-1, 1))  
>>> y = np.array([5, 20, 14, 32, 22, 38])
```

Now, you have two arrays: the input, x , and the output, y . You should call `.reshape()` on x because this array must be **two-dimensional**, or more precisely, it must have **one column** and **as many rows as necessary**. That's exactly what the argument `(-1, 1)` of `.reshape()` specifies.

This is how x and y look now:

Python



```
>>> x
array([[ 5],
       [15],
       [25],
       [35],
       [45],
       [55]])

>>> y
array([ 5, 20, 14, 32, 22, 38])
```

Step 3: Create a model and fit it

The next step is to create a linear regression model and fit it using the existing data.

Create an instance of the class `LinearRegression`, which will represent the regression model:

Python



```
>>> model = LinearRegression()
```

This statement creates the [variable](#) `model` as an instance of `LinearRegression`. You can provide several optional parameters to `LinearRegression`:

- **`fit_intercept`** is a [Boolean](#) that, if `True`, decides to calculate the intercept b_0 or, if `False`, considers it equal to zero. It defaults to `True`.

- **normalize** is a Boolean that, if True, decides to normalize the input variables. It defaults to False, in which case it doesn't normalize the input variables.
- **copy_X** is a Boolean that decides whether to copy (True) or overwrite the input variables (False). It's True by default.
 - **n_jobs** is either an integer or None. It represents the number of jobs used in parallel computation. It defaults to None, which usually means one job. -1 means to use all available processors.

Your `model` as defined above uses the default values of all parameters.

It's time to start using the model. First, you need to call `.fit()` on `model`:

Python



```
>>> model.fit(x, y)
LinearRegression()
```

With `.fit()`, you calculate the optimal values of the weights b_0 and b_1 , using the existing input and output, `x` and `y`, as the arguments. In other words, `.fit()` **fits the model**. It returns `self`, which is the variable `model` itself. That's why you can replace the last two statements with this one:

Python



```
>>> model = LinearRegression().fit(x, y)
```

This statement does the same thing as the previous two. It's just shorter.

