# STA 712 Challenge Assignment 5: Logistic regression in Python

**Due:** Wednesday, October 26, 12:00pm (noon) on Canvas.

**Instructions:**

- Submit your work as a single typed PDF (you should not need to type much, if any, math on this assignment).
- You are welcome to work with others on this assignment, but you must submit your own work.
- You can probably find the answers to many of these questions online. It is ok to use online resources! And using online documentation and examples is a very important part of coding.

## R vs. Python for statistics and data science

Our language of choice in this class has been R, which is a common and popular choice for fitting and working with statistical models. R is particularly good for many core statistical tools: there is excellent support for linear models (and variants like weighted regression and robust regression), GLMs, GAMs, mixed effects models, etc. Through `tidyverse` packages like `tidyr`, `dplyr`, and `ggplot`, R is also a good choice for data cleaning, manipulation, and visualization.

Python is another language which is becoming increasingly popular for data science and machine learning. The `scikit-learn` module contains a wide variety of tools for fitting prediction models like regressions, support vector machines, and random forests. An advantage of `scikit-learn` is that all models have a similar structure: you can fit them using the `.fit()` function, you can get predicted probabilities with the `.predict_proba()` function, etc.

Whether you use R or Python (or SAS, or SPSS, or Stata, etc.) ultimately depends on a combination of personal preferences and the task at hand. The purpose of this challenge assignment is to introduce you to fitting models in Python. Do we need Python to fit logistic regression? No – R is pretty great at this. But it is valuable to see how Python works (and how it behaves differently to R). In the process, you will also see the general procedure for fitting a model using `scikit-learn`, and you will be briefly introduced to other important Python modules like `numpy`, `pandas`, and `scipy`.

## Set up

To complete this challenge assignment, you will need to install Python on your computer. If you do not already have Python installed (or even if you do!) I recommend installing the Anaconda distribution (`https://www.anaconda.com/products/distribution`). You will also need to install the following modules:

- `pandas`
- `numpy`
- `scikit-learn`
- `scipy`

- `matplotlib`
- `statsmodels`

If you install Anaconda, all of these except `statsmodels` should already be included. To install `statsmodels`, see the instructions at `https://www.statsmodels.org/stable/install.html`.

Once Python is installed, how do you use it? If you have the latest versions of R and RStudio installed, you can actually use Python in RStudio! RStudio supports Quarto documents (these are one of the options when you create a new document in RStudio), which behave similarly to RMarkdown documents. In a Quarto document you can include chunks of Python code; see `https://quarto.org/docs/computations/python.html` to get started.

## Logistic regression in Python

In the following questions, we will replicate parts of the analysis on the SBA data from HW 4, Question 3. *Note: I have provided some scaffolded questions here to guide your analysis, but you may still need to research* how *to actually do some of these steps. E.g.,* "how to create a new column in pandas".

1. At the beginning of your document (e.g., in a Python chunk at the top of your Quarto file), import all the required modules.
2. Load the SBA data into Python, using the `pandas.read_csv` function.
3. List the variables in the SBA data that you used to answer the research questions in HW 4.
4. Using the `MIS_Status` column, create a *new* column in your SBA data called `Default`, which is equal to 1 if the loan was charged off (i.e., the borrower defaulted), and 0 if the loan was paid in full (the borrower did not default).
5. Create a *new* column in your SBA data called `Amount` which is the *transformed* loan amount. Use whichever transformation (e.g., log) you used in HW 4 to fix the shape assumption.
6. In R, categorical variables automatically get converted to indicator variables when we fit a logistic regression model. This is not true in Python; part of our data pre-processing is to create the indicator variables we need. This can be done with the `sklearn.preprocessing.OneHotEncoder` class. Create a new dataset called `sba_encoded` which contains your `Amount` column from Question 5, and one-hot encodings of `UrbanRural` and `NewExist`. (For the purposes of this activity, we will ignore any potential interactions between the explanatory variables). *Hint: you will probably want to use* `drop = 'first'` *in your one-hot encoding!*
7. Using the `sklearn.linear_model.LogisticRegression` class, the `Default` column from Question 4, and the `sba_encoded` data from Question 6, fit a logistic regression model and report the estimated coefficients. *Hint: you will want to use* `penalty = 'none'` *when creating the model.*
8. Do the estimated coefficients from Question 7 agree with the estimated coefficients for the same model in R? How do your estimated coefficients change when you change the `solver` in your logistic regression?
9. Using the `sklearn.metrics.log_loss` function, calculate the deviance for your logistic regression model in Python, and compare to the deviance reported by R.

10. Using your fitted model in Python, perform a hypothesis test to address the first research question from HW 4: Is there a relationship between loan amount and the probability the business defaults on the loan, after accounting for whether or not the business is new, and whether it is in an urban or rural environment?

11. As you can see from the previous questions, the `scikit-learn` module is very good for building and assessing prediction models, but is less useful for doing statistical *inference*. For example, we don't get a nice summary table for our model with estimated standard errors, we need to calculate deviance separately, etc.

    One way to get these nice summaries in Python is with the `statsmodels` module. Using the `statsmodels.GLM` class, fit the same logistic regression model as above. Use the `.summary()` function to report a nice table with the estimated coefficients and standard errors. *Hint: make sure to add an intercept column to the* `sba_encoded` *data. The* `statsmodels` *module does not include an intercept for you.*

12. Explain why the standard errors for the `NewExist` and `Intercept` coefficients are so high. How would we fix that issue?

13. Finally, let's try some regression diagnostics. Python has less support for logistic regression diagnostics than R, so we will have to write functions for these diagnostics ourselves. For simplicity, we'll just focus on making a quantile residual plot.

    Write a function to generate quantile residuals for your fitted model in Question 7. You should be able to adapt your code from HW 2; use `numpy.random.uniform` to sample from a uniform distribution, and `scipy.stats.norm.ppf` for the inverse CDF of a standard normal.

14. Using your function from Question 13, create a quantile residual plot for `Amount`. The `matplotlib.pyplot.scatter` function will be useful for creating a scatterplot in Python.