Web Application Using Flask Framework

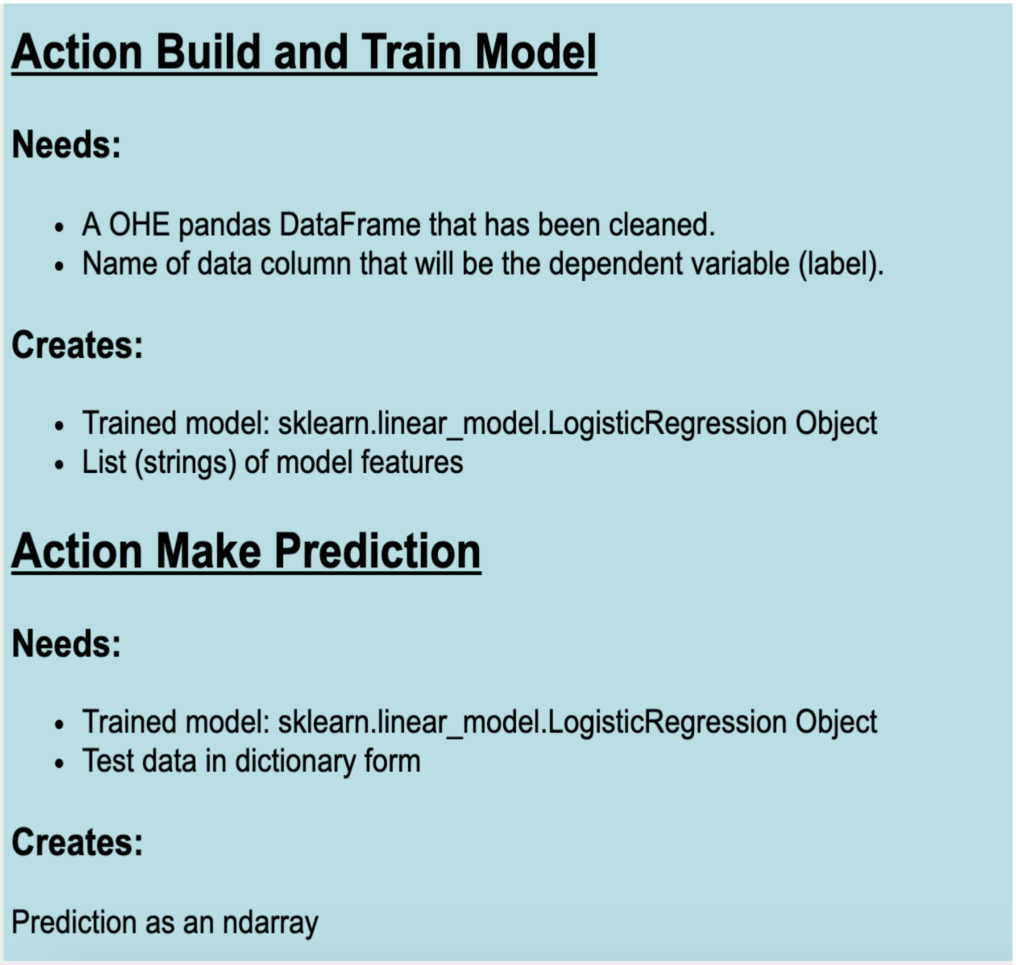
A Few Minor Adjustments

The last sentence of the previous page stated: “*the Data Scientist can just continue to write her/his code as before with only **minor adjustments**.*”. In this page we will discuss those “minor adjustments”.

First we need to understand the simple protocol that software developers use to facilitate communication between the web application and its components (eg. DS Model code). Since the DS Model module is decoupled, it has to have some way to receive from the web application the features and labels that it will use in the model. We do not want to hard-code in those values since that would make the module inflexible. Also, the module needs some way of returning the trained model as well as a prediction based on prediction data that the user provides.

So we will make use of a protocol that software developers use to solve the problem of communication with a decoupled module. The protocol is called “Application Program Interface”, or simply API. *Implementation of API’s does not involve new technology.* Implementation of API’s is simply arranging your code so that it conforms to a simple standard. In fact, for our demo purposes we can say that our API’s will just be functions or methods of a class that can be called by API consumers. (For the purpose of brevity, we will use the term function to mean either a function or a class method.) These API’s will have a documented list of parameters that are passed to the function and a documented list of values that the function will return.

Before we dive into code examples, let's go through the process of API design. The path to writing reliable API's must include a knowledge of what the API's need to do their job and what information that they return. A short summary of API requirements for the demo web application looks like:



Action Build and Train Model

Needs:

- A OHE pandas DataFrame that has been cleaned.
- Name of data column that will be the dependent variable (label).

Creates:

- Trained model: sklearn.linear_model.LogisticRegression Object
- List (strings) of model features

Action Make Prediction

Needs:

- Trained model: sklearn.linear_model.LogisticRegression Object
- Test data in dictionary form

Creates:

Prediction as an ndarray

In the document snippet on the right we see the specifications for two API's. The Data Scientist working on this project probably has code that:

1) Builds and trains the DS model

2) Makes a prediction with the DS model

So this document implies that the Data Scientist must work his/her code so that it contains two functions.

The first function will be called in order to, obviously, build and train the DS model. This API will require a pandas DataFrame that has been One Hot Encoded (OHE)and the column name of the model's label. The model will return both a trained model in the form of a LogisticRegression object together with a list of feature names that have been OHE.

The second function API will be called in order to calculate predictions based on user supplied data. So this function API will require the trained model which is a LogisticRegression object, and will return the predictions in the form of an ndarray.