**Creating a Machine Learning Pipeline Application**

**Introduction:**

**What we will be Covering in this Section**

* Packaging and structuring the code we saw on Section 4

**What we will be covering:**

* Work within a Github repo
* Introduce Tooling (such as Tox)
* Adding Prediction Module
* Introduce Testing
* Versioning
* Logging
* Packaging

**Why does this matter?**

* Reproducibility, which includes:
  + Dependencies
  + Version of the training data
  + Model configuration
  + Which feature have been prepared in the model
* Reduce the risk of error by applying tests

**Training the model**

* Rough Package Structure
  + <https://realpython.com/python-application-layouts/#installable-single-package>
* setenv =
  + PYTHONPATH=.
  + (That where the “.” Means current directory
* (run *tox* in *regression\_model*)
  + Notice that *.tox* folder is then created, which is where tox installs it’s virtual environments
* (use *tox -r* to rerun tox environment creation)

**Creating the Pipeline**

* 6.3
* Show config\config.py
* Show trained\_models\pipeline.py
* Show trained\_models\train\_pipeline.py
* Show tox.ini
* Show that now by executing *tox* we are actually persisting a model

**Making predictions with the model**

* Show requirements.txt
  + Pytest was added
* Show \processing\data\_managements
* Show \trained\_models\predict
* Show \tests
  + We are loading the whole dataset
  + Grabbing the first row ordered by records (reproducibility / remove randomness)
  + Test 1: Check if the prediction is not null
  + Test 2: Check if the prediction is a float
  + Test 3: Check if the rounded prediction is == 112476
* Show tox.ini
  + Pytest was added
    - Runs all the tests in the test directory
* Use tox -r to rebuild the environment (as Pytest was added)

**Data Validation in the Model Package**

* Show trained\_models\predict (validate\_inputs)
* Show processing\validation
* Show tests\test\_predict

**Feature Engineering in the Pipeline**

* Show processing\features
* Show trained\_models\pipeline

**Versioning and Logging**

* Versioning is important to know the which inputs, transformation and algorithms went into generating a prediction at a particular point in time
* Logging is important to capture the events in our system which allows us to debug complex sequences as well as to provide a record of what our system has done. This can be useful both for internal and external audit as well as potential regulatory requirements
* Show VERSION
* Show trained\_models\\_\_init\_\_
  + Here we name our logger as \_\_name\_\_ (\_\_name\_\_ = “regression\_model”
  + Set to logging.DEBUG which is expecting when we are still developing
    - In production it would be INFO
  + We then add a handler (defined in config\logging\_config
* Show config\logging\_config
  + The logging.Formatter is defined to be
    - A datetime stamp
    - The name of the logger
    - The log level, one of: debug, info, warning or error
    - The function name: the function which the lock was generated and the
    - line number within that function
    - The message that we logged
  + get\_console\_handler
    - Assigns the handler from the logging stream handler which is going to be the system standard output (which is either the command prompt or the terminal). A file could be another alternative
    - We then apply the formats defined above
    - We then return the handler
* Show processing\data\_management
  + Notice the three new imports
  + Notice that we using the getLogger to get the logger that we have already created
  + Notice in save\_file\_name that now we persist our pickle file along with the \_version
  + Notice the remove\_old\_pipelines where:
    - We loop over all the files in our model directory and we delete everything apart from a few files to keep
* Show pocessing\erros which allows us to specify some specific errors (for good practice)
* Show processing\features\ which now has the InvalidModelInputError
* When running tox:
  + Notice more information:
    - Datetime
    - Which module
    - Which function
    - The actual log message (containing the version)
* Show trained\_models\lasso\_regresion\_output\_v0.1.0.pkl
* Show trained\_models\predict
  + In this case, you should also save the log in a file and/or database (which we don’t)
* By saving the log correctly we are able to look back and know which inputs we used, which version of the system was in use (including the model version) as well as the outcome predictions
* Show test\tox, if we want to include the log we insert -s in “pytest -s tests”
* Show trained\_models\train\_pipeline, show that you can replace ‘\_\_name\_\_’ by ‘regression\_model’

**Building the Package**

Packaging: grouping the files

* Show requirements
  + *Setuptools*: Improvements to Python *distutils* (which is an older library) that facilitates building and distributing Python packages
  + *Wheel*: Official binary distribution format for Python provides:
    - An extension to *setuptools*
    - A command line to working with *wheel* files
      * A *wheel* file is a zip format archive
* Show setup
* Show tox.ini
  + [testenv:install\_locally]
    - Has the same dependencies as the ones we used before
  + We do train the pipeline first
  + But now we run python setup.py
    - Sdist is to create the source distribution
    - Bdist\_wheel is to create the wheel distribution
  + We need to run tox using -e parameter to run the package
* Run tox -e install\_locally
* Show dist folder
  + Notice the wheel file
  + And source distribution file (tar.gz)
  + We use wheel first and it will fall back to the source distribution files if there’s a problem with the wheel. So it’s good to have both
* Don’t worry about the contents of the *build* nor *regression\_model.egg-info* directory
* Latter on we will learn how we can distribute this created package
  + But now, we are able to install it locally
* Run *pip intall -e* in regression\_model folder
  + -e is for local packages
  + Then we need to specify the root directory of our package, but as we are in the package’s folder already, “.” will suffice
* Run *python*
* Run *import regression\_model*
* Run regression\_model.\_\_version\_\_
* Show the MANIFEST file
  + It specifies the files to include in the package and those to exclude
  + Change the paths if you want to do it for your own packages

**Wrap Up**

* Versioning applies not only to the model, but to the data and dependencies as well
* We organized preprocessing and features engineering steps into a pipeline
* We persisted predictions, including inputs and outputs, through logs.