

# PIPELINES

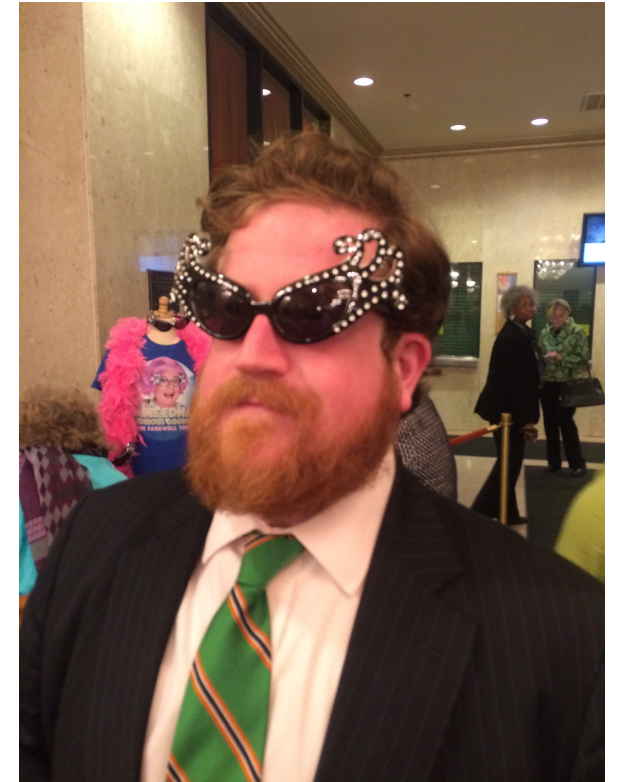
Charles Rice, Journeyman Data Scientist

---

# WHO THE HELL AM I?

---

- DSI-2 Alumnus
- Recovering speechwriter, journalist, and editor
- Belshazzar's Feast (Daniel 5:26)
- DS Interests: NLP, NLG, Debunking motivational posters
- Non-DS Interests:
  - World history to 1945, dead languages, musical theater, and the Oxford Comma



---

# AGENDA

---

- ▶ What is a Pipeline?
- ▶ Pipeline Transformations
- ▶ Why Use a Pipeline?
- ▶ Coding Implementation
- ▶ Sklearn documentation

---

# WHAT IS A PIPELINE?

---

- ▶ In the ‘real world’ what is a pipeline?



---

## WHAT IS A PIPELINE?

---



Prospects

Inquiries

Proposals/Sales

New Customers

---

## WHAT IS A PIPELINE?

---

- ▶ Using the ‘real world’ examples, then, what might a pipeline be for programming in general, and data science in particular?

---

## WHAT IS A PIPELINE?

---

Technical definition:

Data pipelines are concatenations of an arbitrary number of sequential transformers and estimators in which the output of one step becomes the input of the next step.

---

## WHAT IS A PIPELINE?

---

Data pipelines are concatenations of sequential transformers and estimators in which the output of one step becomes the input of the next step.

In the abstract, they look something like this:





---

# PIPELINE TRANSFORMATIONS

---


Examples of data transformations?

---

# PIPELINE TRANSFORMATIONS

---

Examples of data transformations?

- ▶ Change in units (lbs  kgs) ('Normalization')
- ▶ Change of scale
- ▶ Change of base
- ▶ Missing data imputation
- ▶ Text/image/sound vectorization!

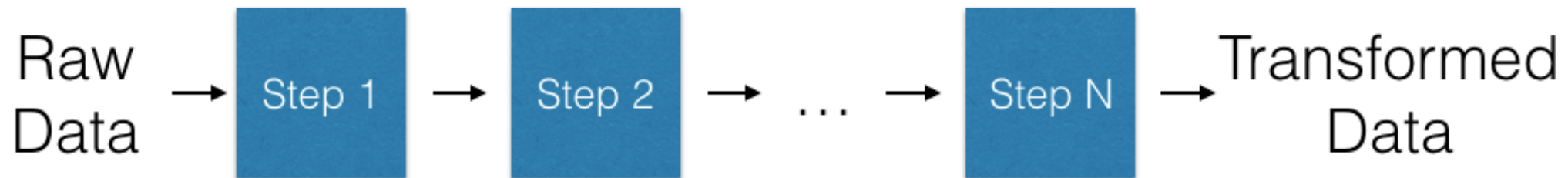
---

## WHAT IS A PIPELINE? (SUMMARY)

---

‘Pipeline’ is, ultimately, some new jargon for something we’ve already used: functions

- ▶ They take input data, run it through an arbitrary number of transformers, and finally through an estimator or classifier (e.g., a logistic regression) to produce the model or predictions



---

## PIPELINE USE CASE

---

You may have noticed (or you will soon notice) that when you get to modeling, you tend to type the same pieces of code over and over and over again, as you tweak parameters

This seems to be especially true with natural language processing.

---

## WHY USE A PIPELINE?

---

- ▶ The primary reasons are efficiency and reproducibility.
- ▶ The pipeline's purpose is to make it possible to set the parameters of multiple transformers and an estimator in one place
- ▶ But their all-in-one nature also means they are uniquely useful for working with text data.

---

## PIPELINE USE CASE

---

Repetition leads to Mistakes. Mistakes lead to Bad Results. Bad Results lead to the Dark Side.

--Yoda (paraphrased)



---

## PIPELINE USE CASE

---

### Debunking Quotations

- Processing about 5 million words
- Blocks of 1000, 500, 100 observations at the sentence, word, and character level
- Many ways to process and, more vexingly, to tweak the processes
- What to do?

---

# PIPELINE USE CASE

---

I wrote a function. Several of them.

```
In [36]: def vec_test(vect, X, y, pair, vec):
          i = 0
          res = []
          accl = []
          precl = []
          featl = []
          while i <= 4:
              X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.3) #random_state=42)
              X_train_dtm = vect.fit_transform(X_train)
              #X_vec = vect.fit_transform(X)
              feat = X_train_dtm.shape[1]
              #print 'Features: ', feat
              featl.append(feat)
              X_test_dtm = vect.transform(X_test)
              nb = MultinomialNB()
              nb.fit(X_train_dtm, y_train)
              y_pred_class = nb.predict(X_test_dtm)
              acc = accuracy_score(y_test, y_pred_class)
              prec = precision_score(y_test, y_pred_class, average='weighted')
              accl.append(acc)
              precl.append(prec)
              #print 'Accuracy: ', acc
```



---

## PIPELINE USE CASE

---

All of that, and I had to keep track of a number of variations, hard coding them.

Whereas a pipeline, which does the core of that function, is just:

```
>>> from sklearn.pipeline import Pipeline
>>> text_clf = Pipeline([('vect', CountVectorizer()),
...                      ('tfidf', TfidfTransformer()),
...                      ('clf', MultinomialNB()),
...                      ])
```

Which can then be fit to data and scored like any other model

---

## PIPELINE USE CASE

---

- ▶ Pipelines are also the only way to tune text vectorizers using gridsearch
- ▶ Which definitely beats testing all the parameters by hand, even if it did threaten to melt my computer
- ▶ Only resource you really need: <http://scikit-learn.org/stable/modules/pipeline.html#pipeline>