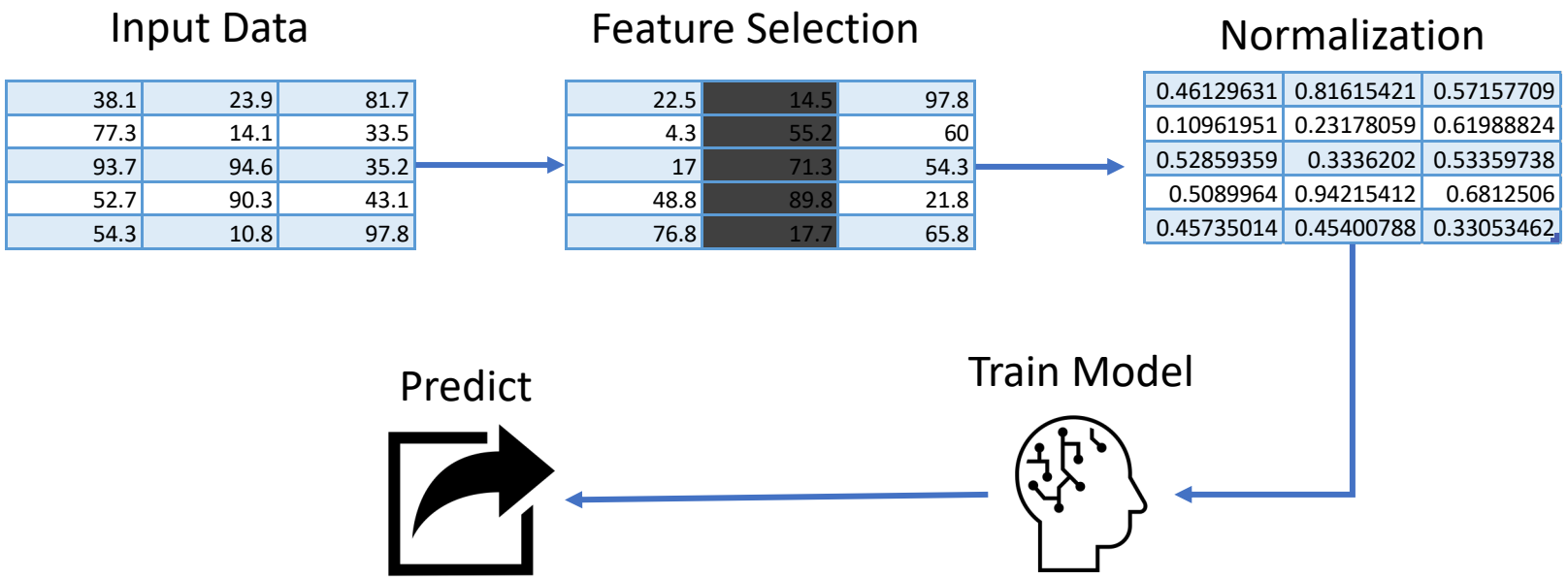
Model Selection

Pipeline Building for ML

ML Pipelines

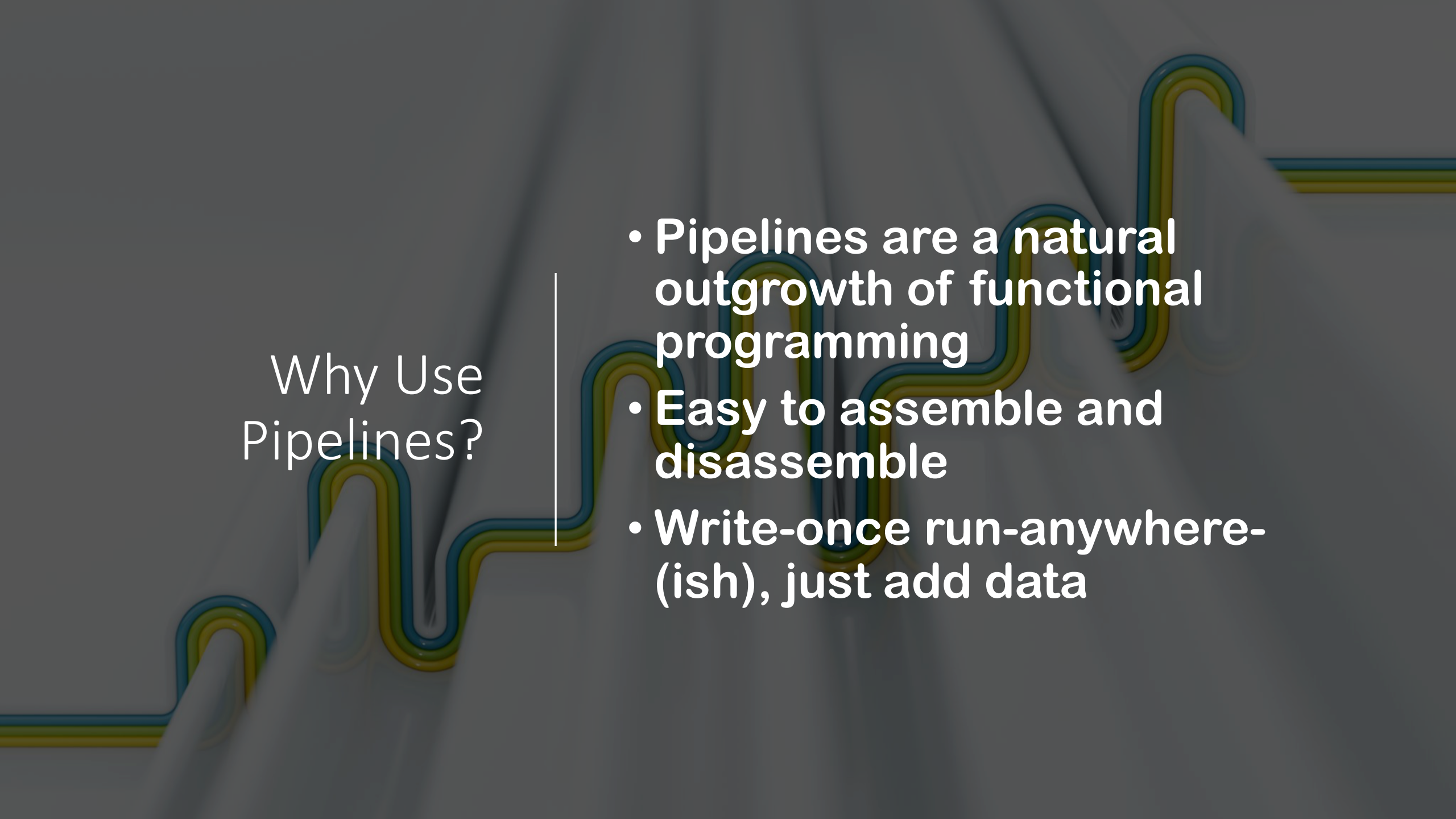
A pipeline is a series of steps which each perform a transformation on the data



What does a Pipeline Look Like?

- **Nothing special: but each function takes the output of the previous step as its input**
- **I can re-run the entire experiment using a different dataset by changing the file-path in the first step**

```
 9  def load_data(path):
10      return pd.read_csv(path)
11
12  def select_features(df):
13      dropped = df.drop(0, 3)
14      return dropped
15
16  def normalize(df):
17      normalized = (df - df.mean())/df.std()
18      return normalized
19
20
21  df = load_data('Users/jack/Data/input.csv')
22  df = select_features(df)
23  df = normalize(df)
```



Why Use Pipelines?

- **Pipelines are a natural outgrowth of functional programming**
- **Easy to assemble and disassemble**
- **Write-once run-anywhere-(ish), just add data**

What Pipelines do we Use?

Prediction Pipeline

Evaluation Pipeline

Training Pipeline

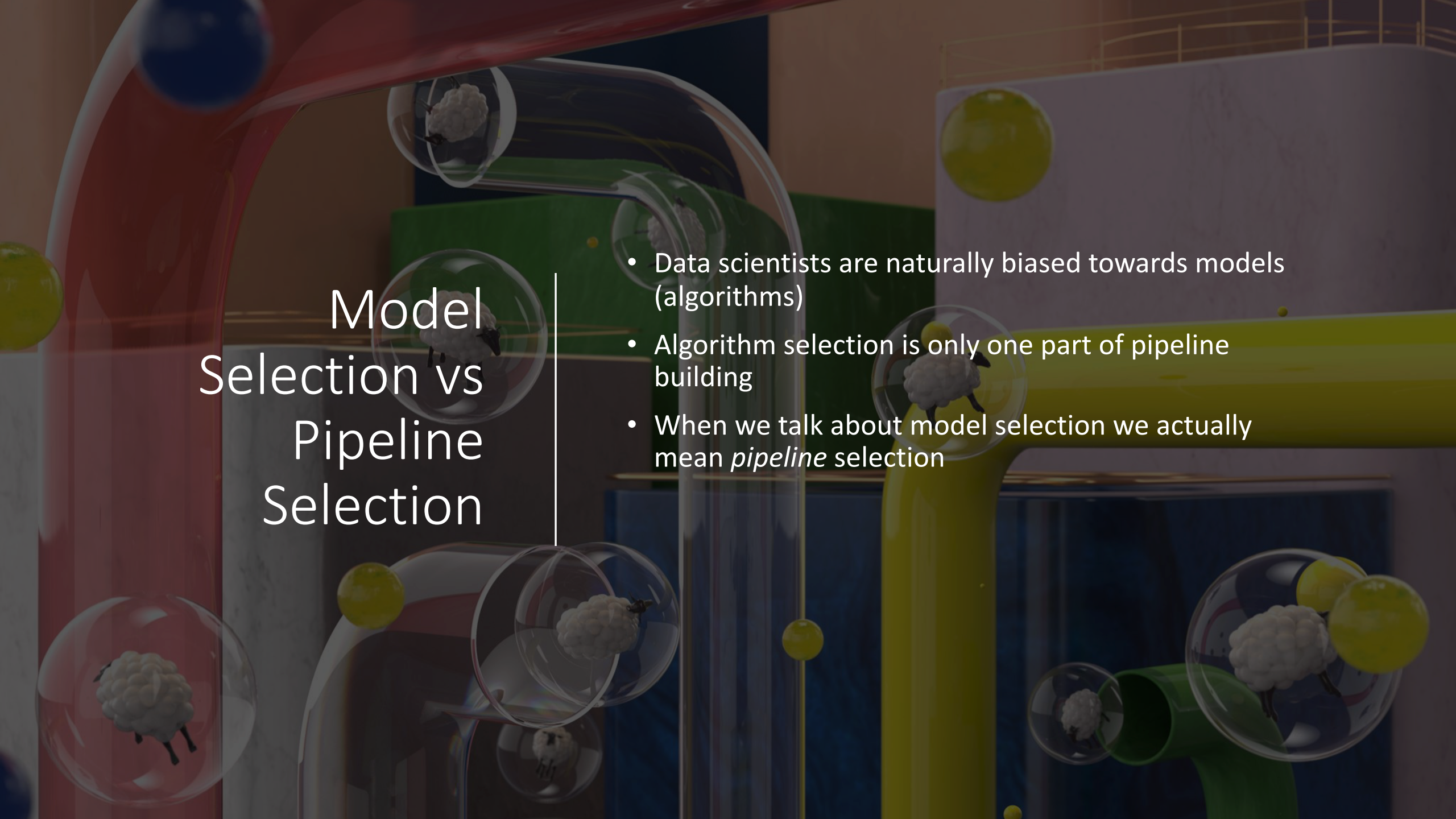
What Pipelines do we Use?

Prediction Pipeline: Takes unlabelled data, produces prediction

Training Pipeline: Takes labelled data, produces predictive model

Evaluation Pipeline: Takes labelled data and candidate pipelines, produces evaluation metrics (accuracy etc.)

Ceci n'est pas une pipe



Model Selection vs Pipeline Selection

- Data scientists are naturally biased towards models (algorithms)
- Algorithm selection is only one part of pipeline building
- When we talk about model selection we actually mean *pipeline* selection

Pipeline Responsibilities

Training Pipeline

- **Fit** any data pre-processing transformations
- Adjust ***model parameters*** through training

Prediction Pipeline

- **Apply** any data pre-processing transformations
- Feed transformed input data to model

Evaluation Pipeline

- Determine which pre-processing should be used
- Adjust ***model hyperparameters*** through evaluation

Label Leakage: Data Imputation

Replace with mean for column

```
imp = SimpleImputer(missing_values=np.nan,
                    strategy='mean')
imp.fit(X)
Xi = imp.transform(X)
```

Impute from similar examples

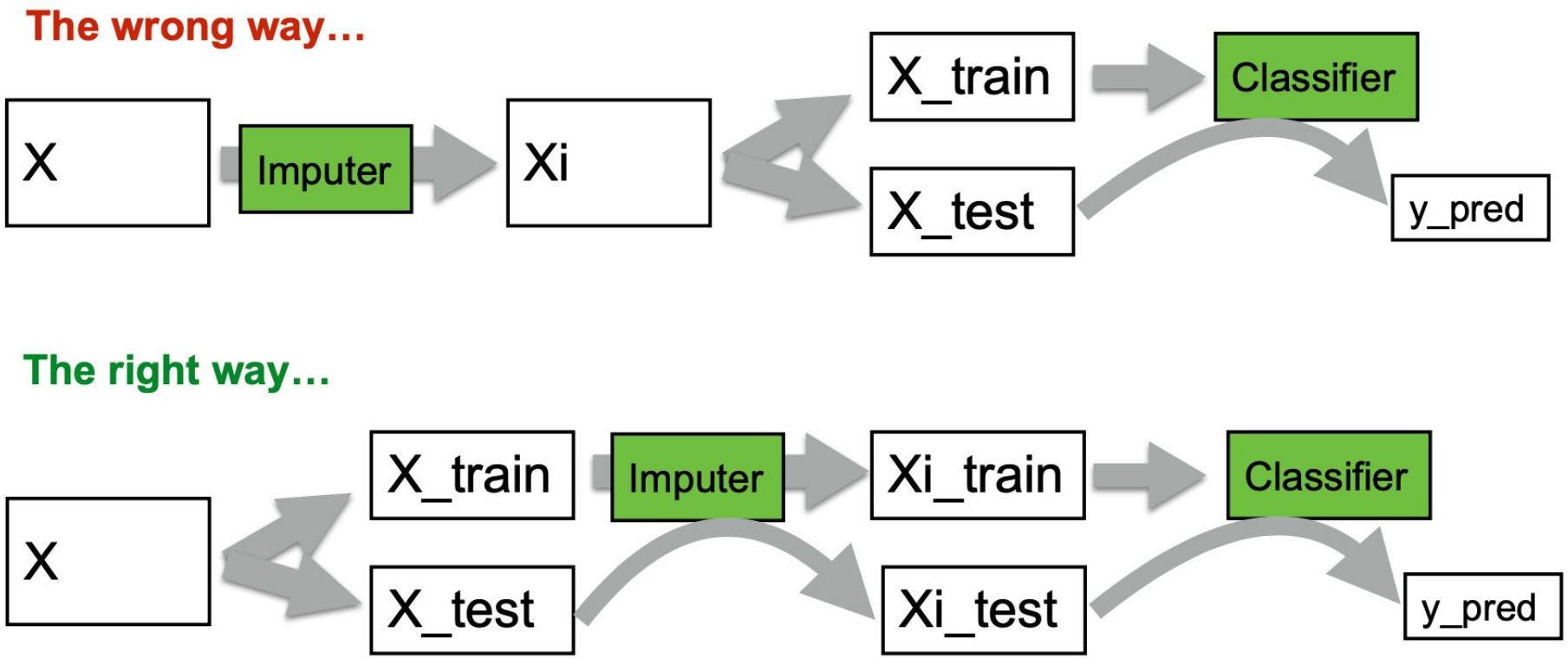
```
imp_kNN = KNNImputer(missing_values = np.nan)
imp_kNN.fit(X)
Xi = imp_kNN.transform(X)
```

Imputer should not have access to test data

UCI Mammographic Mass Data

Age	Shape	Margin	Density	Severity
67.0	3.0	5.0	3.0	1
43.0	1.0	1.0	NaN	1
58.0	4.0	5.0	3.0	1
28.0	1.0	1.0	3.0	0
74.0	1.0	5.0	NaN	1
65.0	1.0	NaN	3.0	0
70.0	NaN	NaN	3.0	0
42.0	1.0	NaN	3.0	0
57.0	1.0	5.0	3.0	1
60.0	NaN	5.0	1.0	1

Preprocessing in the Evaluation Pipeline



Parameters and Hyperparameters

Parameters

- Estimated by the learning algorithm
  - Coefficients in linear models
  - Weights in neural net
  - Conditional probabilities in Naïve Bayes
  - Support Vectors in SVM

Hyperparameters

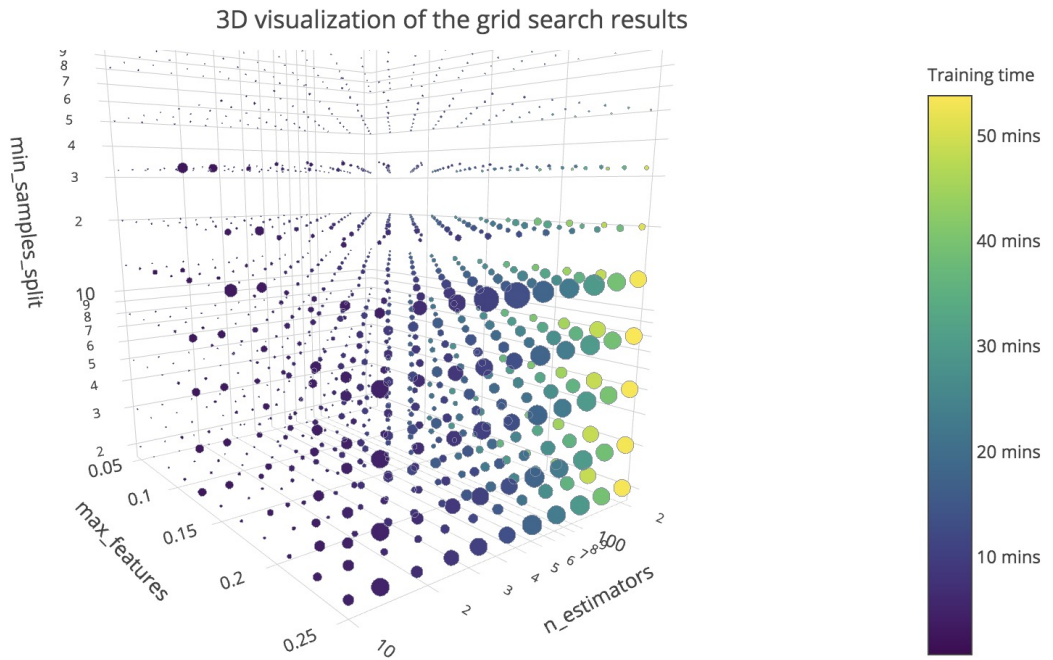
- Set by hand
  - k in k nearest neighbour
  - max depth in decision tree
  - split criterion in decision tree
  - α in gradient descent

In practice: hyper parameter tuning might be automated

Does this not make them regular parameters?

(Hyper-)Parameter Tuning

- Very few hard-and-fast rules for which hyper-parameters to choose for any algorithm / pre-processing step
- **Grid search** is often used to brute force the problem



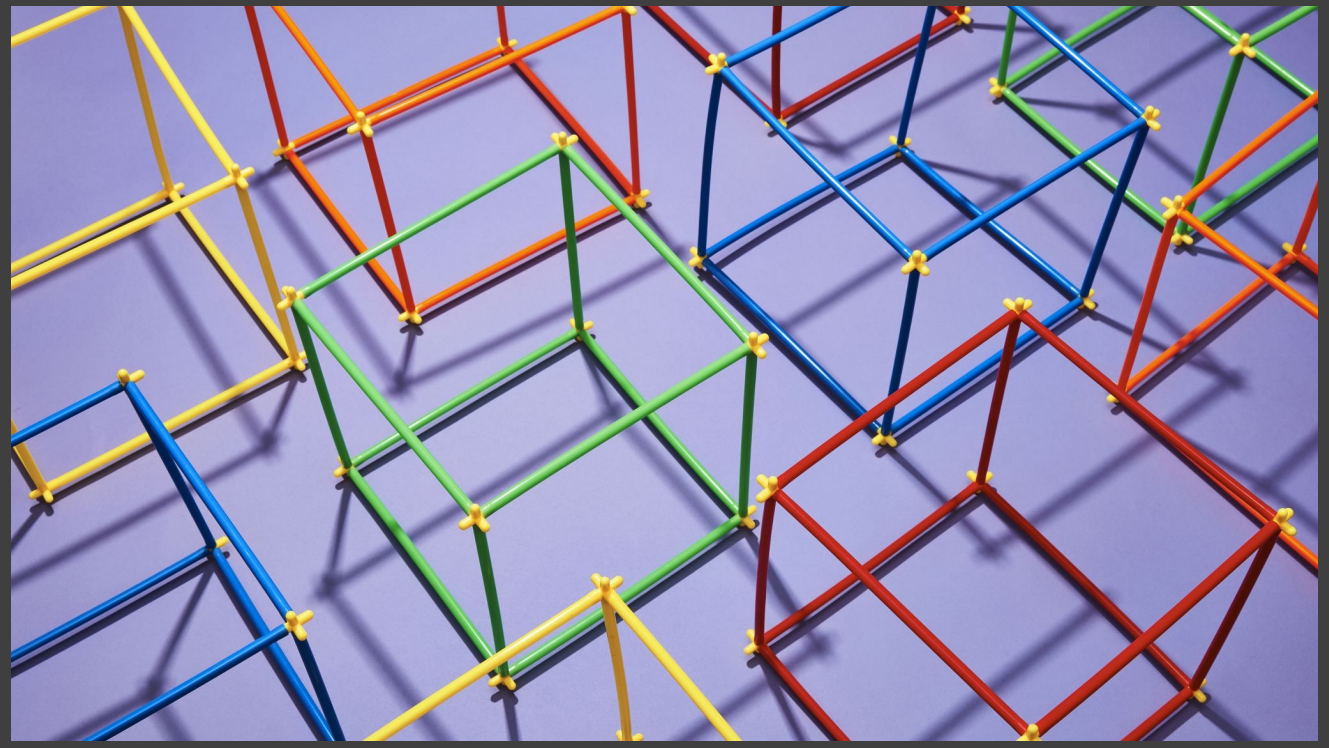
Grid Search and Hyperparameter Tuning

- The grid is the space of every possible combination of hyperparameters
- KNeighborsClassifier
  - n_neighbors: {1, 3, 5, 10}
  - weights: {'uniform', 'distance'}
  - metric: {'euclidean', 'manhattan'}

4 x 2 x 2 = 16 combinations

Grows exponentially with number of hyperparameters

Pipelines: Scikit Learn



- Scikit Learn has an implementation of pipelines which can be useful for performing grid search
- Usually consists of one or more pre-processing steps followed by a final model-building step
- Each pre-processing step must implement **fit** and **transform**
- The model-building step must implement **fit**

Pipeline: Hold-out Testing

- Pipeline
  - **Two Transforms**
    - KNNImputer
    - StandardScaler
  - **One Estimator**
    - KNeighborsClassifier

```
kNNpipe  = Pipeline(steps=[
    ('imputer', KNNImputer(missing_values = np.nan)),
    ('scaler', StandardScaler()),
    ('classifier', KNeighborsClassifier())])
In [150]:
kNNpipe.fit(X_train, y_train)
y_pred = kNNpipe.predict(X_test)
print("Accuracy: {0:4.2f}".format(accuracy_score(y_test,y_pred)))
confusion_matrix(y_test, y_pred)
```

Pipeline: Cross-Validation

- Pipeline object passed to **cross_val_score**
- All fitting and transforming done automatically
  - New imputer and scaler at each fold

```
kNNpipe  = Pipeline(steps=[
    ('imputer', KNNImputer(missing_values = np.nan)),
    ('scaler', StandardScaler()),
    ('classifier', KNeighborsClassifier())])


acc_arr = cross_val_score(kNNpipe, X, y, cv=5)
print("Accuracy: {0:4.2f}".format(sum(acc_arr)/len(acc_arr)))
```

- Hold-out accuracy 0.82
- Cross-Val accuracy 0.78

Why the difference? Which is more reliable?

Pipeline: Grid Search

We can use a pipeline to automate hyperparameter grid-search for a model algorithm with SKLearn

```
knn = KNeighborsClassifier()

param_grid = {'n_neighbors':[1,3,5,10],
              'metric':['manhattan','euclidean'],
             'weights':['uniform','distance']}

knn_gs = GridSearchCV(knn, param_grid, cv=10,
                      verbose = 1, n_jobs = -1)
```

Parameter sets are ‘scored’ based on the default score for the classifier (for KneighborsClassifier this is accuracy)

Using GridSearchCV

- We call the **fit** method on GridSearchCV to find the best combination of parameters for a given dataset
- Can run evaluation in parallel for significant speed increases

```
knn = KNeighborsClassifier()

param_grid = {'n_neighbors':[1,3,5,10],
              'metric':['manhattan','euclidean'],
             'weights':['uniform','distance']}
In [16]:
knn_gs = GridSearchCV(knn, param_grid, cv=10,
                      verbose = 1, n_jobs = -1)
knn_gs = knn_gs.fit(X_trainS,y_train)

Fitting 10 folds for each of 16 candidates, totalling 160 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  68 tasks      | elapsed:    1.7s
[Parallel(n_jobs=-1)]: Done 160 out of 160 | elapsed:    2.1s finished
```

Using GridSearchCV

- The GridSearchCV object is a classifier

```
y_pred_gs = knn_gs.predict(X_testS)
```

- Explicitly build a classifier with the best parameters

```
knn_gs.best_params_
Out[25]:
{'metric': 'manhattan', 'n_neighbors': 1, 'weights': 'uniform'}
In [19]:
knn2 = KNeighborsClassifier(metric= 'manhattan',
                            n_neighbors = 1, weights = 'uniform')
```

- Unpack the best parameter directly

```
knn3 = KNeighborsClassifier(**knn_gs.best_params_)
```

RandomizedSearchCV

- A randomised rather than exhaustive search
- Suitable when parameter space is huge
- Parameter Search Budget can be Set
- Insensitive to parameters that don't matter

Model Parameters vs Pre-processing

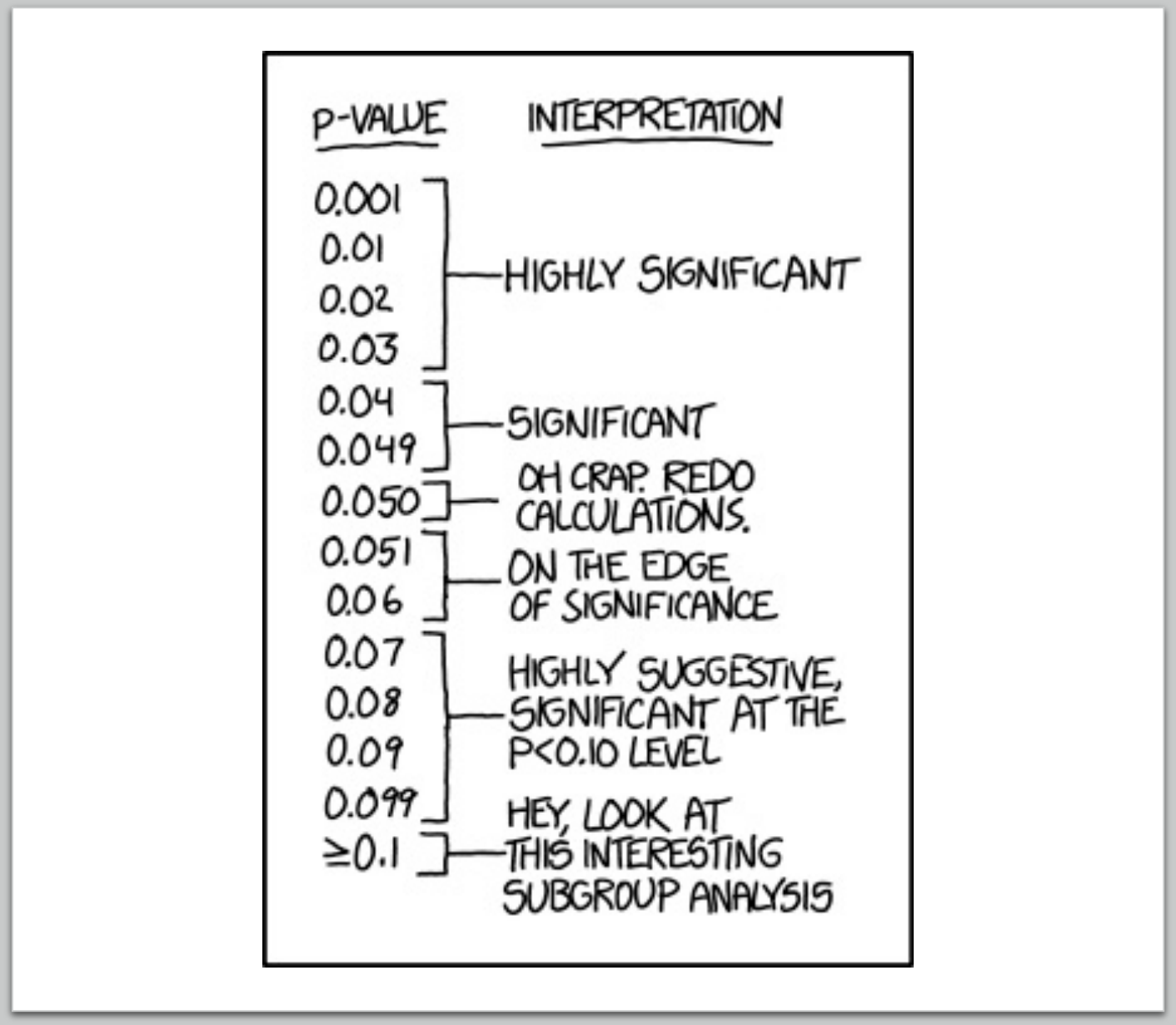
- We're not building models, we're building pipelines
- GridsearchCV focuses on algorithm hyperparameters
- Pre-processing is part of our training pipeline so should be included
- Not fully supported but can be done manually

A Note on Generalizability

- Hyperparameter tuning finds the best hyperparameter set for the *current training data*
- I could perform a grid search to find the best random seed for training
- What's important is how well our pipeline performs *after deployment*

Good Enough?

- How sure do we need to be about our chosen pipeline's expected performance?
- If we're deploying a system for commercial use we often don't need cast-iron guarantees
- If we're making a contribution to the field of data science then we really do



Statistical Comparisons of Classifiers over Multiple Data Sets

Janez Demsar

Excellent paper outlining best practice for best practice in evaluating ML pipelines academically

- Never average results across datasets
- Avoid tests with parametric assumptions
  - Don't use T-Tests, use Wilcoxon Signed Rank Tests
  - Don't use ANOVAs, use Friedman tests
- Use more datasets!
  - Generally require > 10 datasets and > 5 algorithms for statistical tests

Summary

Pipelines

- Training Pipelines, Prediction Pipelines, Evaluation Pipelines
- Data Pre-processing and label leakage
- Sklearn Pipelines make our lives easier

Summary

Grid Search

- Hyperparameters can be brute-forced
- Exponential running time
- Is it worth it?
- Randomized Search Alternative