# mlinspect: Inspect ML Pipelines in Python in the form of a DAG

Stefan Grafberger

Technical University Munich

November 2, 2020

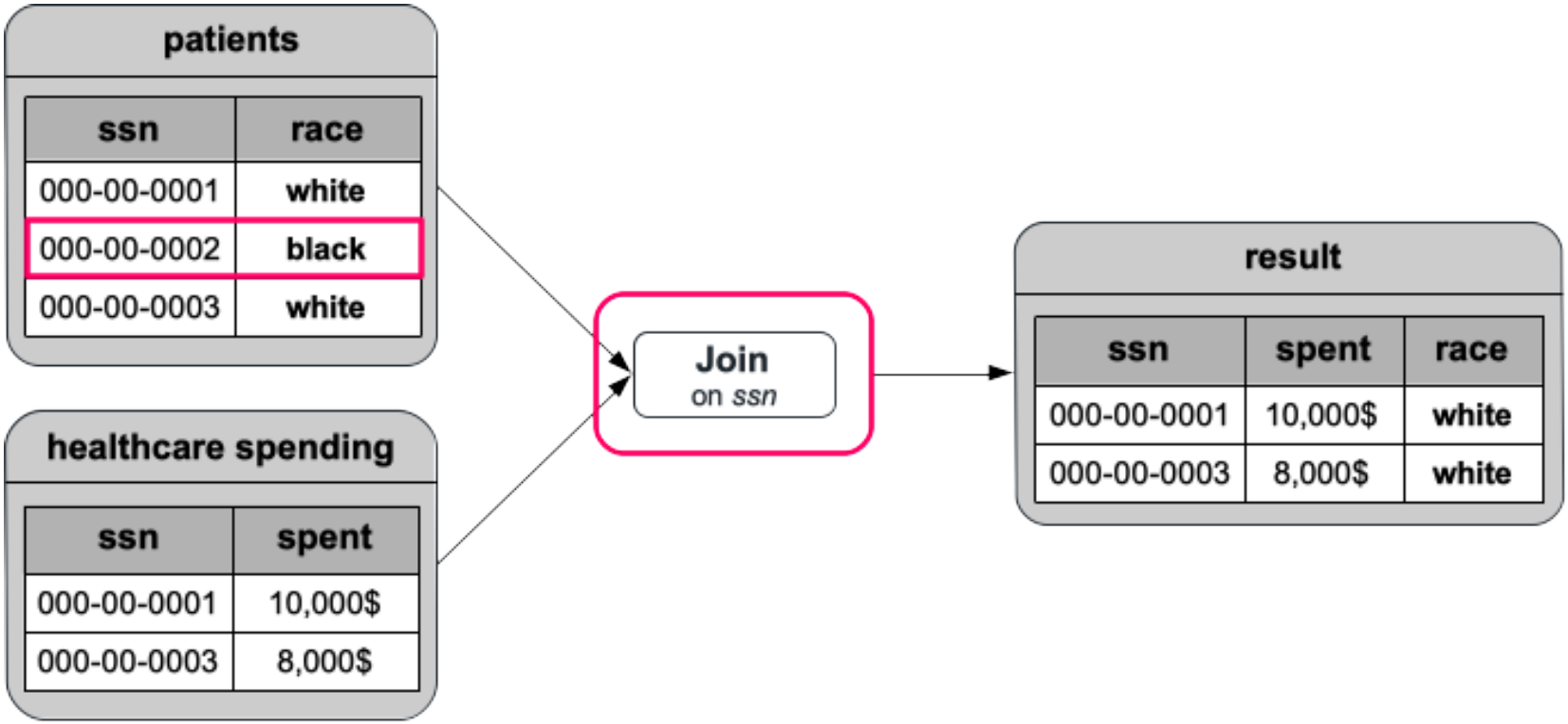
# Motivation

- ML is everywhere and is increasingly used to automate decisions that impact peoples' lives, some examples:
  - loan applications
  - medical diagnosis
  - job applications
- ML Systems can be very brittle and there are many problems with the fairness, transparency and accountability of these systems
- Issues can be hard to detect without manual and time-intensive scrutiny

# Issue Sources

- Data
  - ⇒ Data quality issues, as well as data over- and under-representation, can heavily impact the results
- ML Model
  - ⇒ Lots of research from the ML community on this, but mostly focuses on the adaption of learning algorithms on static datasets
- Preprocessing code
  - Input data for ML applications has to be integrated, preprocessed and cleaned first
  - Creating a good ML preprocessing pipeline is difficult and requires a lot of ML expertise
  - ⇒ We need to look at the pipeline as a whole, not just at data and model!

# Example




Operations like Selections, Joins and Missing Value imputation can introduce data distribution issues

# mlinspect

⇒ Similar to code inspections in modern IDEs

- Can automatically detect issues like statistical bias and provides linting for best-practices
- Can also help with debugging
- Works on existing pipeline code using popular libraries without requiring modifications
- Negligible performance overhead

# Scikit-learn Pipeline Example

**Potential issues in preprocessing pipeline:**

1. Join might change proportions of groups in data
2. Column 'age_group' projected out, but required for fairness
3. Selection might change proportions of groups in data
4. Imputation might change proportions of groups in data
5. 'race' as a feature might be illegal!
6. Embedding vectors may not be available for rare names!

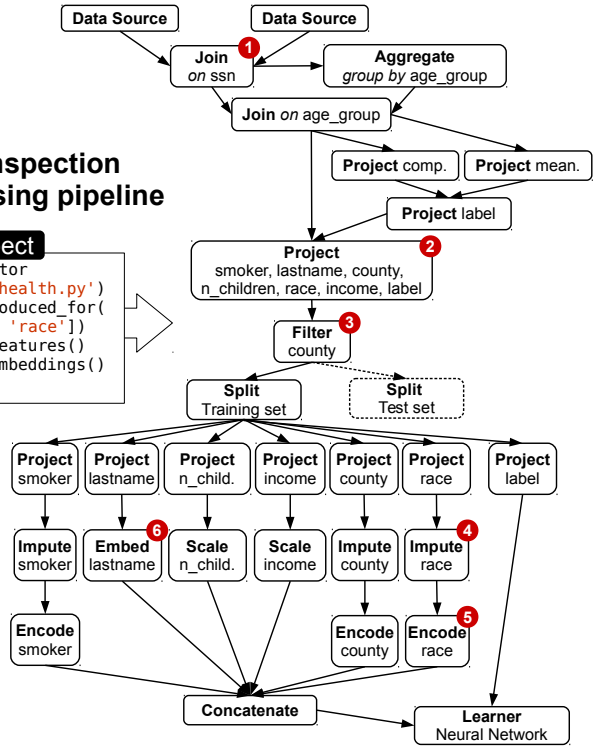
**Python script for preprocessing, written exclusively with native pandas and sklearn constructs**

```
# load input data sources, join to single table
patients = pandas.read_csv(…)
histories = pandas.read_csv(…)
data = pandas.merge([patients, histories], on=['ssn'])

# compute mean complications per age group, append as column
complications = data.groupby('age_group')
  .agg(mean_complications=('complications','mean'))
data = data.merge(complications, on=['age_group'])

# Target variable: people with frequent complications
data['label'] = data['complications'] >
  1.2 * data['mean_complications']

# Project data to subset of attributes, filter by counties
data = data[['smoker', 'last_name', 'county',
             'num_children', 'race', 'income', 'label']]
data = data[data['county'].isin(counties_of_interest)]

# Define a nested feature encoding pipeline for the data
impute_and_encode = sklearn.Pipeline([
  (sklearn.SimpleImputer(strategy='most_frequent')),
  (sklearn.OneHotEncoder())])
featurisation = sklearn.ColumnTransformer(transformers=[
  (impute_and_encode, ['smoker', 'county', 'race']),
  (Word2VecTransformer(), 'last_name')
  (sklearn.StandardScaler(), ['num_children', 'income']])

# Define the training pipeline for the model
neural_net = sklearn.KerasClassifier(build_fn=create_model())
pipeline = sklearn.Pipeline([
  ('features', featurisation),
  ('learning_algorithm', neural_net)])

# Train-test split, model training and evaluation
train_data, test_data = train_test_split(data)
model = pipeline.fit(train_data, train_data.label)
print(model.score(test_data, test_data.label))
```

**Declarative inspection of preprocessing pipeline**

```
mlinspect
PipelineInspector
.on_pipeline('health.py')
 .no_bias_introduced_for(
 ['age_group', 'race'])
 .no_illegal_features()
 .no_missing_embeddings()
.verify()
```

**Corresponding dataflow DAG for instrumentation, extracted by *mlinspect***




Example of an ML pipeline that predicts which patients are at a higher risk of serious complications.