Program Synthesis for Data Science

Rohan Bavishi

rbavishi@cs.berkeley.edu

Berkeley
UNIVERSITY OF CALIFORNIA

Data Science

Harvard Business Review

Data Scientist: The Sexiest Job of the 21st Century

Meet the people who can coax treasure out of messy, unstructured data. by Thomas H. Davenport and D.J. Patil

Why Choose Data Science for Your Career

Rinu Gour Apr 20, 2019 · 6 min read ★

Bright and Auspicious Future of Data Science – Learn it Before you Regret

Published on May 3, 2021

Data Science

Courtesy: Data Science Life Cycle, Chanin Nantasenamat

Data Science

Platforms

H2O.ai

DataRobot

Google's AutoML

Azure Machine Learning

Programming Libraries / APIs

pandas

NumPy

TensorFlow

matplotlib

seaborn

scikit learn

plotly

PyTorch

Data Collection → Data Cleaning → Data Exploration → Model Building → Deployment

Courtesy: Data Science Life Cycle, Chanin Nantasenamat

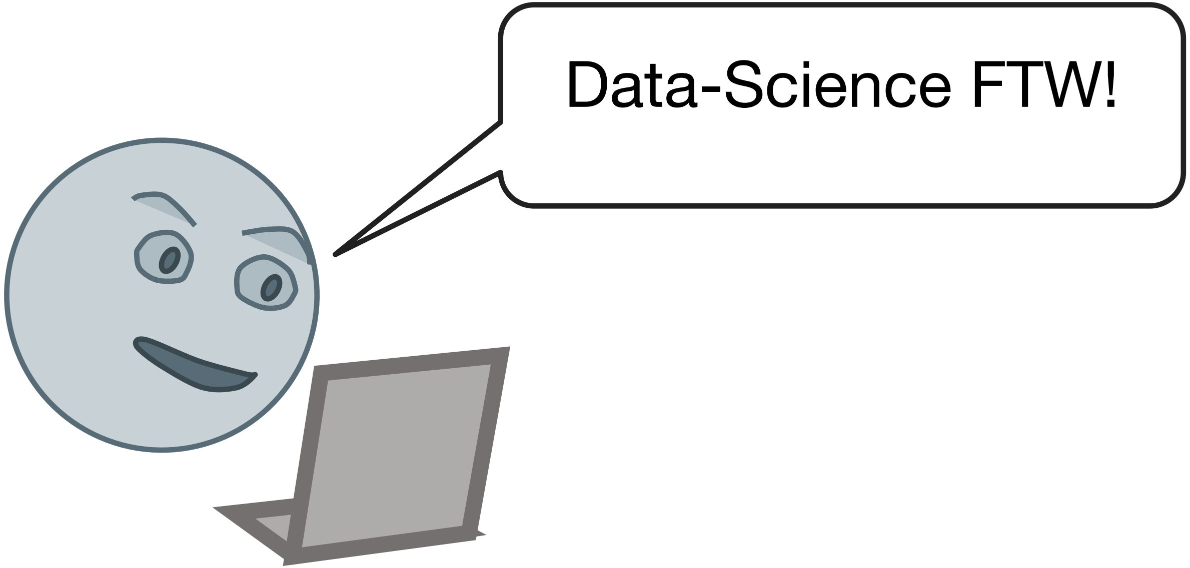
The Problem

Using these APIs can be... Frustrating

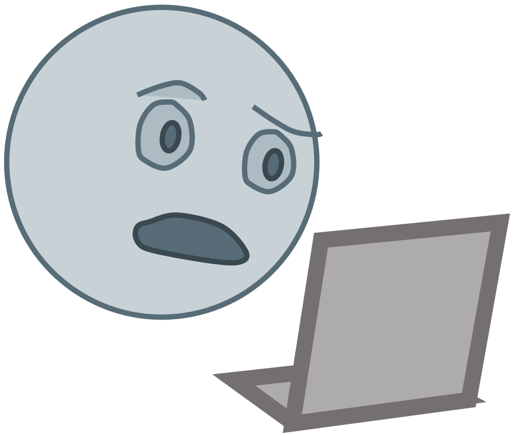
- Large and Complex
- Dense Documentation
- Lack of Sufficient Examples within Documentation

Steep Learning Curve

Case in Point: Pandas



Case in Point: Pandas



DataFrame

Constructor

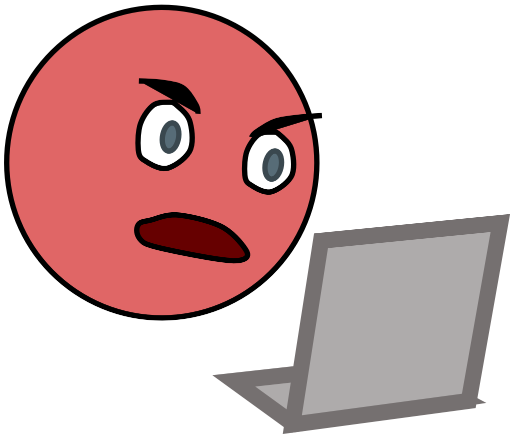
`DataFrame([data, index, columns, dtype, copy])`	Two-dimensional size-mutable, potentially heterogeneous tabular data structure with labeled axes (rows and columns).

Attributes and underlying data

Axes

`DataFrame.index`	The index (row labels) of the DataFrame.
`DataFrame.columns`	The column labels of the DataFrame.
`DataFrame.dtypes`	Return the dtypes in the DataFrame.
`DataFrame.ftypes`	(DEPRECATED) Return the ftypes (indication of sparse/dense and dtype) in DataFrame.
`DataFrame.get_dtype_counts(self)`	(DEPRECATED) Return counts of unique dtypes in this object.
`DataFrame.get_ftype_counts(self)`	(DEPRECATED) Return counts of unique ftypes in this object.
`DataFrame.select_dtypes(self[, include, exclude])`	Return a subset of the DataFrame's columns based on the column dtypes.
`DataFrame.values`	Return a Numpy representation of the DataFrame.
`DataFrame.get_values(self)`	(DEPRECATED) Return an ndarray after converting sparse values to dense.
`DataFrame.axes`	Return a list representing the axes of the DataFrame.
`DataFrame.ndim`	Return an int representing the number of axes / array

Case in Point: Pandas



pandas.DataFrame.pivot_table

`DataFrame.pivot_table`(*self, values=None, index=None, columns=None, aggfunc='mean', fill_value=None, margins=False, dropna=True, margins_name='All', observed=False*) [source]

Create a spreadsheet-style pivot table as a DataFrame. The levels in the pivot table will be stored in MultiIndex objects (hierarchical indexes) on the index and columns of the result DataFrame.

Parameters:

values : *column to aggregate, optional*

index : *column, Grouper, array, or list of the previous*

If an array is passed, it must be the same length as the data. The list can contain any of the other types (except list). Keys to group by on the pivot table index. If an array is passed, it is being used as the same manner as column values.

columns : *column, Grouper, array, or list of the previous*

If an array is passed, it must be the same length as the data. The list can contain any of the other types (except list). Keys to group by on the pivot table column. If an array is passed, it is being used as the same manner as column values.

aggfunc : *function, list of functions, dict, default numpy.mean*

If list of functions passed, the resulting pivot table will have hierarchical columns whose top level are the function names (inferred from the function objects themselves) If dict is passed, the key is column to aggregate and value is function or list of functions

fill_value : *scalar, default None*

Value to replace missing values with

margins : *boolean, default False*

Add all row / columns (e.g. for subtotal / grand totals)

dropna : *boolean, default True*

Do not include columns whose entries are all NaN

margins_name : *string, default 'All'*

Name of the row / column that will contain the totals when margins is True.

observed : *boolean, default False*

This only applies if any of the groupers are Categoricals. If True: only show observed values for categorical groupers. If False: show all values for categorical groupers.

Changed in version 0.25.0.

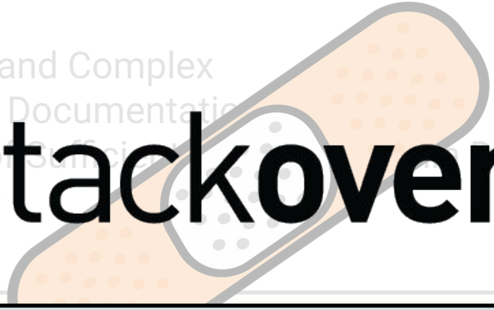
Returns: DataFrame

Using these APIs can be… Frustrating

- Large and Complex
- Dense Documentation
- Lack of Sufficient Examples within Documentation

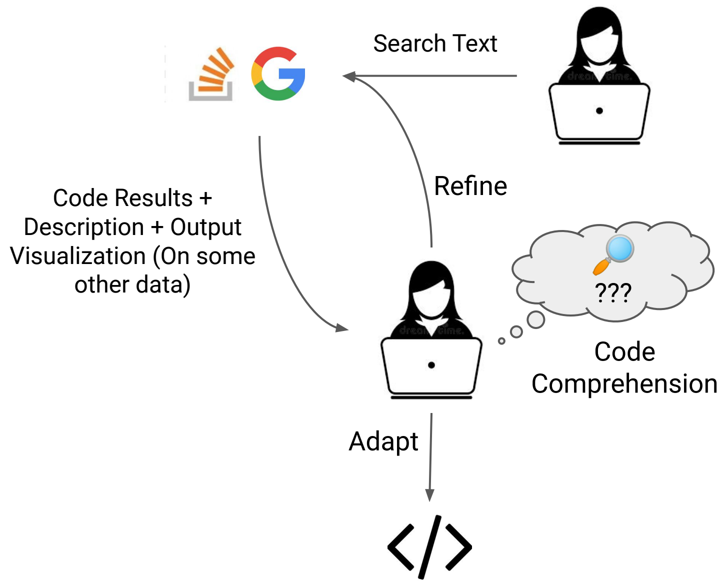
Steep Learning Curve

The Current Remedy

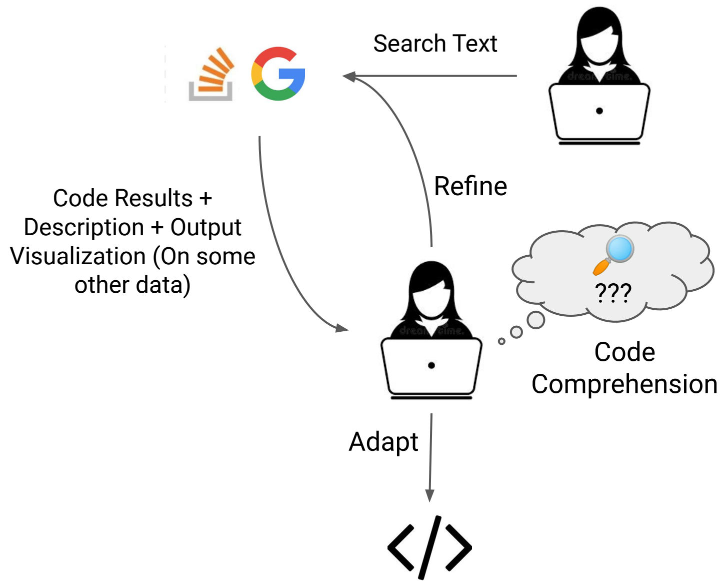


StackOverflow serves as a rich treasure trove of **reusable** examples

Reusing Code from StackOverflow - A Model

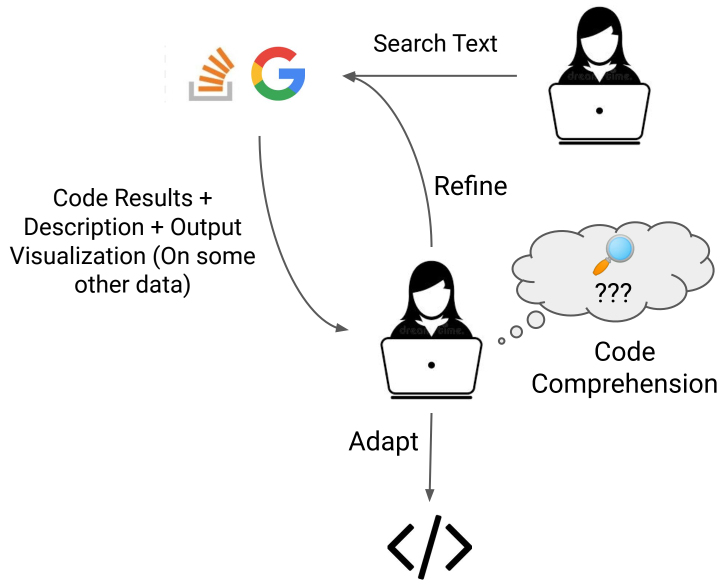


Problems with Reusing StackOverflow Code



All three -
Searching, Comprehension and Adaptation
are ***non-trivial***

Problems with Reusing StackOverflow Code



Searching is Non-Trivial

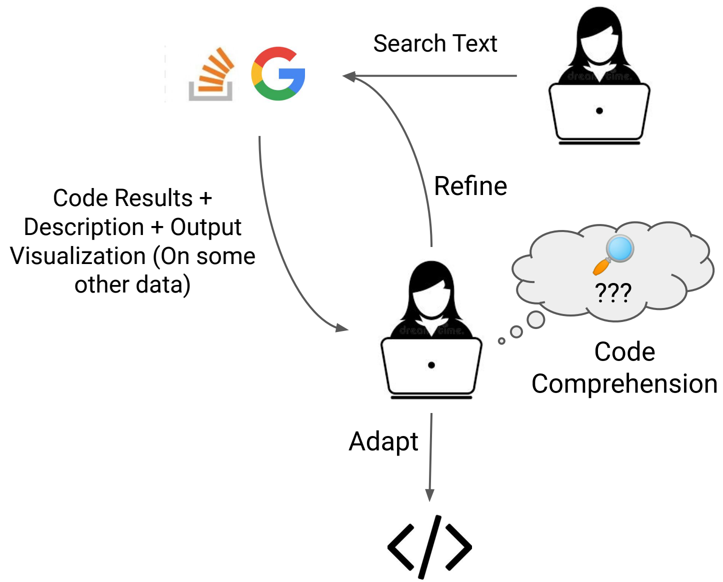
Queries for reusable code snippets frequent but difficult to write [1]

User Comments reported in [1]:

- *"Because it's hard to explain in a sentence what the code snippet you're looking for should do"*

[1] What do developers search for on the web?, Xia et al. Empirical Software Engineering 2016

Problems with Reusing StackOverflow Code



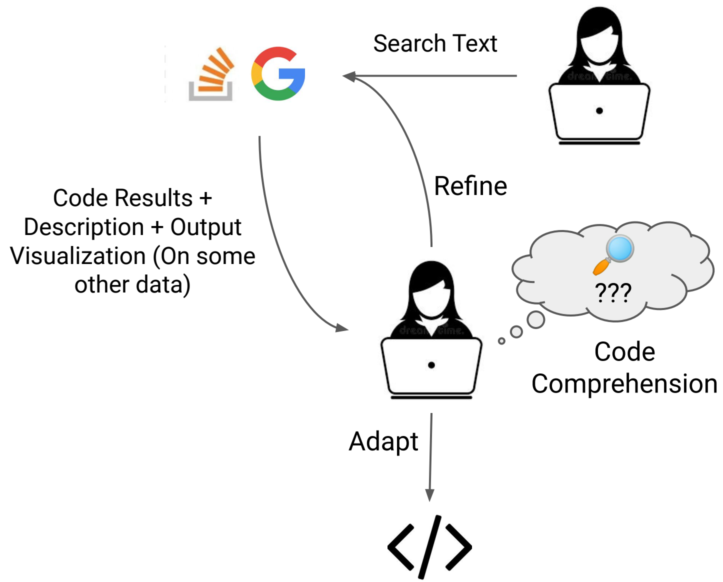
Code Comprehension is Non-Trivial

What is the code doing?

Is this doing what I want to do with my own input?

Are there any assumptions? Do they hold in my case?

Problems with Reusing StackOverflow Code

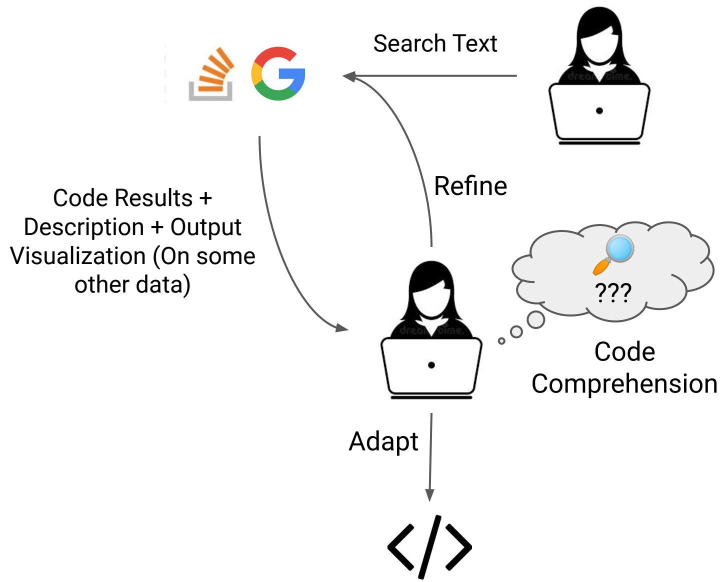


Code Comprehension is Non-Trivial

Incomprehensible code one of the reasons behind less reuse [2]

[2] How do developers utilize source code from stack overflow, Wu et al. Empirical Software Engineering 2019

Problems with Reusing StackOverflow Code

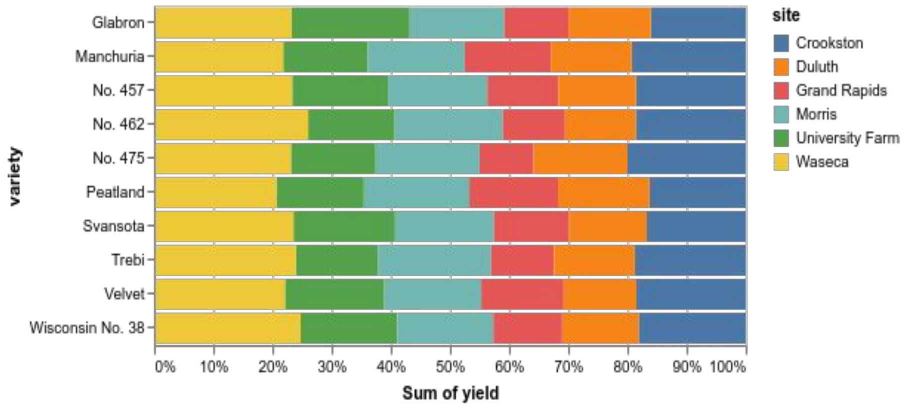


Adaptation is Non-Trivial

- Adaptation overhead reduces reuse [2]
  - Code needs to be modified in about 78.2% of cases

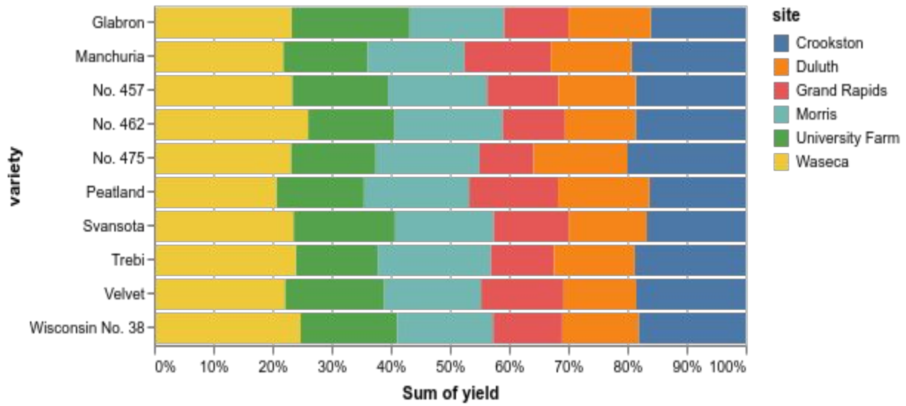
[2] How do developers utilize source code from stack overflow, Wu et al. Empirical Software Engineering 2019

Problems with StackOverflow: An Example



Say you want to make a plot like this
(Distribution of a variable for each distinct value of another variable)
(A Normalized Stacked Bar Chart)

Problems with StackOverflow: An Example



Say you want to make a plot like this
(Distribution of a variable for each distinct value of another variable)

Searching

Search for “normalized stacked bar chart” on stackoverflow / Google

Problems with StackOverflow: An Example

https://stackoverflow.com › questions › how-can-i-nor...

How can I normalize data and create a stacked bar chart?

Aug 3, 2019 — Use the **normalization** scale that you want. import **pandas** as pd from sklearn import preprocessing import **matplotlib.pyplot** as plt # Read data ...

2 answers · Top answer: This is probably something that you are trying to achieve. You can use ...

https://stackoverflow.com › questions › normalize-data-...

normalize data and plot as stacked bar plot with python/ pandas

Aug 8, 2018 — I think that the pivot function is what I am looking for. However before I want to **normalize** the "count" **column** for each land_cover relative to ...

2 answers · Top answer: I think you need: df['Count Per Canopy Cat'] = (df['count'] * df['tc_densit...

https://stackoverflow.com › questions › how-to-plot-sta...

How to plot stacked bar-charts in pandastic way?

Oct 7, 2019 — How to **plot stacked bar-charts** in pandastic way? **python pandas matplotlib** group-by bar-**chart**. I'have dataframe mostly with categorical columns:

2 answers · 2 votes: I would do something like this: df1=df.groupby('col_to_group')['col_1','col_2'...

https://stackoverflow.com › questions › how-do-i-make...

How do I make pandas catagorical stacked bar chart scale to ...

May 22, 2019 — I am trying to produce a **stacked bar chart** based on counts of different catagories (the 'Class' column in my dataframe). My data is also grouped ...

1 answer · Top answer: There is a lot you can do with matplotlib to forcibly scale the y axis so t...

https://stackoverflow.com › questions › how-to-plot-sta...

How to plot stacked & normalized histograms?

Oct 6, 2019 — How to **plot stacked** & **normalized** histograms? **python pandas** numpy **matplotlib**. I have a dataset that maps continuous values to discrete categories ...

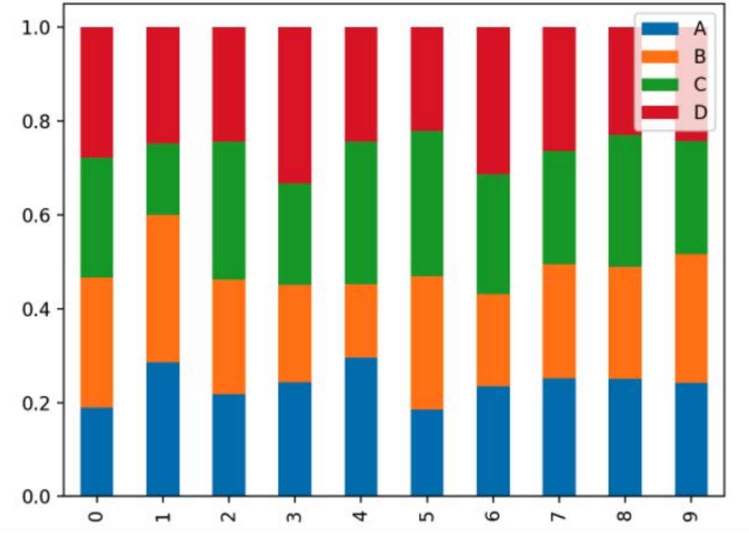
2 answers · 1 vote: I dont know if this is really that much more compact or readable than what y...

Searching

Search for “normalized stacked bar chart” on stackoverflow / Google

The 5th result is the first relevant one

Problems with StackOverflow: An Example



Comprehension / Adaptation

Hard to read code / How to adapt?

```
# One column per category, 1 if maps to category, 0 otherwise
df2 = pd.DataFrame({
        'score' : df.score,
        'A' : (df.category == 'A').astype(float),
        'B' : (df.category == 'B').astype(float),
        'C' : (df.category == 'C').astype(float),
        'D' : (df.category == 'D').astype(float)
    },
    columns=['score', 'A', 'B', 'C', 'D'])

# select "bins" of .1 width, and sum for each category
df3 = pd.DataFrame([df2[(df2.score >= (n/10.0)) & (df2.score < ((n+1)/10.0))].iloc[

# Sum over series for weights
df4 = df3.sum(1)

bars = pd.DataFrame(df3.values / np.tile(df4.values, [4, 1]).transpose(), columns=l

bars.plot.bar(stacked=True)
```

The Opportunity

Synthesis-powered Productivity Tools for Data Scientists
(Low Code / No Code)

where users can:

Easily Specify their **Intent**

Explore solutions at **interactive speeds**

Push-button integration of solutions into their existing workflow

The Opportunity

Synthesis-powered Productivity Tools for Data Scientists
(Low Code / No Code)

where users can:

Easily Specify their **Intent**	Explore solutions at **interactive speeds**	**Push-button integration** of solutions into their existing workflow

My Research

Research

AutoPandas: Neural-Backed Generators for Program Synthesis
***Rohan Bavishi**, Caroline Lemieux, Roy Fox, Koushik Sen, Ion Stoica*
OOPSLA 2019

Gauss

Gauss: Program Synthesis by Reasoning Over Graphs
***Rohan Bavishi**, Caroline Lemieux, Koushik Sen, Ion Stoica*
OOPSLA 2021

VizSmith

VizSmith: Automated Visualization Synthesis by Mining Data-Science Notebooks
***Rohan Bavishi**, Shadaj Laddad, Hiroaki Yoshida, Mukul R. Prasad, Koushik Sen*
ASE 2021

Research

AutoPandas: Neural-Backed Generators for Program Synthesis
***Rohan Bavishi**, Caroline Lemieux, Roy Fox, Koushik Sen, Ion Stoica*
OOPSLA 2019

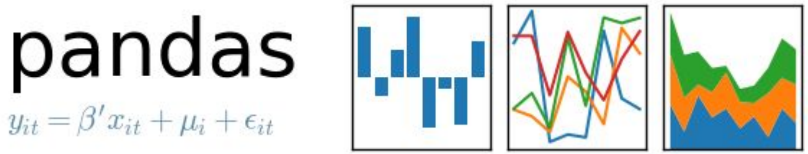
Gauss: Program Synthesis by Reasoning Over Graphs
***Rohan Bavishi**, Caroline Lemieux, Koushik Sen, Ion Stoica*
OOPSLA 2021

VIZSMITH

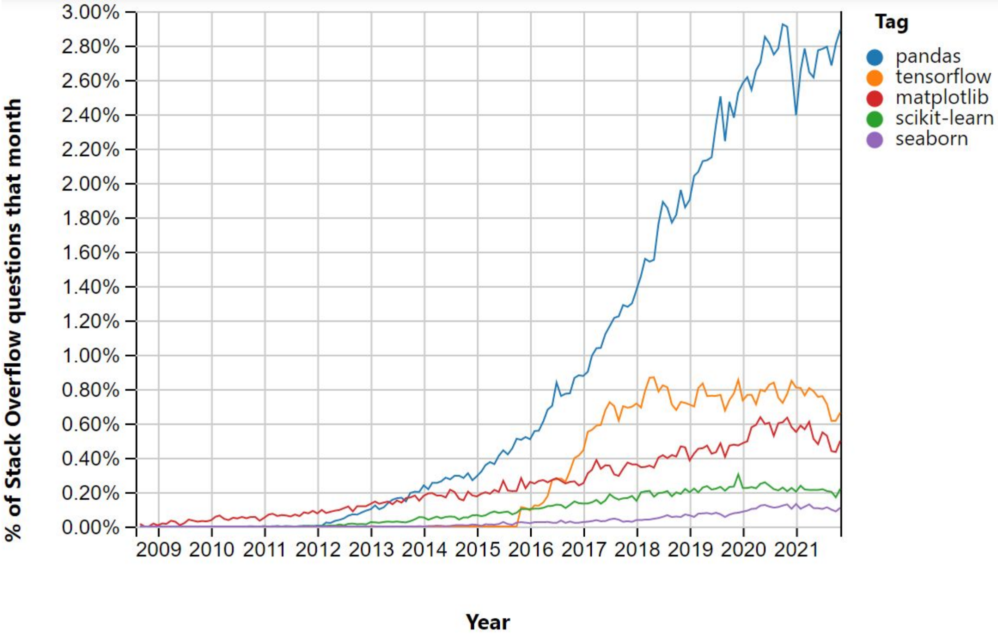
VizSmith: Automated Visualization Synthesis by Mining Data-Science Notebooks
***Rohan Bavishi**, Shadaj Laddad, Hiroaki Yoshida, Mukul R. Prasad, Koushik Sen*
ASE 2021

AutoPandas: Synthesis for Pandas

Why Pandas?

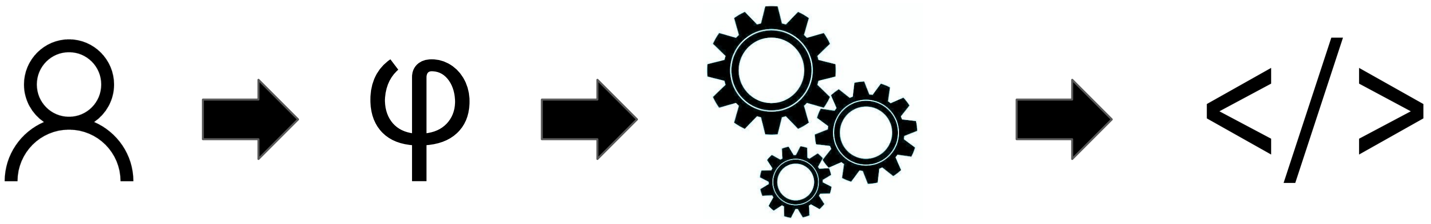


- The most popular Python library for data cleaning and processing
- 2-3% of StackOverflow questions tagged with pandas

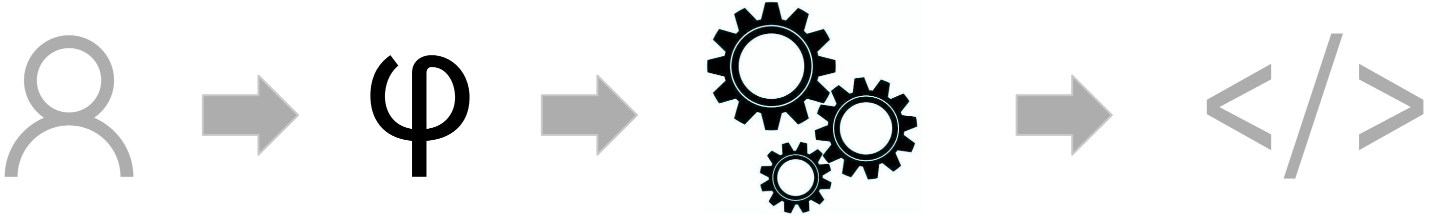


Percentage of StackOverflow questions for different tags

What is Program Synthesis?



Program Synthesis: The Three Dimensions



User

Synthesis Specification / User Intent

Synthesis Engine

Synthesized Program (that satisfies the spec)

*There are **three** main dimensions under our control in Synthesis*
[Program Synthesis Book, Gulwani et al. 2017]

The **Form** of Synthesis Specification / User Intent (Modality)	The Search Space	The Search Algorithm

A Decision Grid for the Three Dimensions

Specification Modality	Formal Specs	Program Sketches	Traces/ Demonstrations	I/O Examples	Natural Language

Search Space	Sketches and Holes (Partial Programs with Holes)	DSLs / Grammars	Anything (Arbitrary Text)

Search Algorithm	Constraint-Solving, CEGIS	Deduction	Enumerative	Stochastic/ ML-Driven
	Commonly Requires Translation to SAT/SMT	Opt-1: Program-Space Opt-2: Data-Space	w/ Pruning Strategy w/ ML Bias	

A Decision Grid for the Three Dimensions

Specification Modality

- Formal Specs
- Program Sketches
- Traces/ Demonstrations
- I/O Examples
- Natural Language

Search Space

- Sketches and Holes (Partial Programs with Holes)

This gives us Program-Sketching (Solar-Lezama)

Search Algorithm

- Constraint-Solving, CEGIS
- Deduction
- Enumerative
- Stochastic/ ML-Driven

Commonly Requires Translation to SAT/SMT

Opt-1: Program-Space
Opt-2: Data-Space

w/ Pruning Strategy
w/ ML Bias

A Decision Grid for the Three Dimensions

Specification Modality

- Formal Specs
- Program Sketches
- Traces/ Demonstrations
- I/O Examples
- Natural Language

This gives us SyGuS (Alur et al.)

Search Space

- Sketches and Holes (Partial Programs with Holes)
- DSLs / Grammars
- Anything (Arbitrary Text)

Search Algorithm

- Constraint-Solving, CEGIS
  Commonly Requires Translation to SAT/SMT
- Deduction
  Opt-1: Program-Space
  Opt-2: Data-Space
- Enumerative
  w/ Pruning Strategy
  w/ ML Bias
- Stochastic/ ML-Driven

A Decision Grid for the Three Dimensions

Specification Modality

- Formal Specs
- Program Sketches
- Traces/ Demonstrations
- **I/O Examples**
- Natural Language

This gives us FlashFill

Search Space

- Sketches and Holes (Partial Programs with Holes)
- **DSLs / Grammars**
- Anything (Arbitrary Text)

Search Algorithm

- Constraint-Solving, CEGIS — Commonly Requires Translation to SAT/SMT
- **Deduction** — Opt-1: Program-Space; Opt-2: Data-Space
- Enumerative — w/ Pruning Strategy; w/ ML Bias
- Stochastic/ ML-Driven

All Three Dimensions Together

Specification Modality

RobustFill, Seq2SQL
- The model is expected to fully produce the program from scratch, in the restricted space.

ns

I/O Examples

Natural Language

Search Space

Sketches and Holes
(Partial Programs with Holes)

DSLs / Grammars

Anything
(Arbitrary Text)

Search Algorithm

Constraint-Solving, CEGIS

Deduction

Enumerative

Stochastic/ ML-Driven

Commonly Requires Translation to SAT/SMT

Opt-1: Program-Space
Opt-2: Data-Space

w/ Pruning Strategy
w/ ML Bias

Tackling Synthesis for Pandas

Specification Modality	Formal Specs	Program Sketches	Traces/ Demonstrations	I/O Examples	Natural Language

Search Space	Sketches and Holes (Partial Programs with Holes)	DSLs / Grammars	Anything (Arbitrary Text)

Search Algorithm	Constraint-Solving, CEGIS	Deduction	Enumerative	Stochastic/ ML-Driven
	Commonly Requires Translation to SAT/SMT	Opt-1: Program-Space Opt-2: Data-Space	w/ Pruning Strategy w/ ML Bias	

Tackling Synthesis for Pandas

We picked I/O examples, because most StackOverflow questions come with an example

Specification Modality
- Formal Specs
- Program Sketches
- Traces/ Demonstrations
- **I/O Examples**
- Natural Language

Search Space
- Sketches and Holes (Partial Programs with Holes)
- DSLs / Grammars
- Anything (Arbitrary Text)

Search Algorithm
- Constraint-Solving, CEGIS — Commonly Requires Translation to SAT/SMT
- Deduction — Opt-1: Program-Space; Opt-2: Data-Space
- Enumerative — w/ Pruning Strategy; w/ ML Bias
- Stochastic/ ML-Driven

Tackling Synthesis for Pandas

We do not want users to have to provide a partial program, nor can we use SAT/SMT. Also arbitrary text seemed too hard :)

Specification Modality

- Formal Specs
- Program Sketches
- Traces/ Demonstrations
- I/O Examples
- Natural Language

Search Space

- Sketches and Holes (Partial Programs with Holes)
- DSLs / Grammars
- Anything (Arbitrary Text)

Search Algorithm

- Constraint-Solving, CEGIS
  Commonly Requires Translation to SAT/SMT
- Deduction
  Opt-1: Program-Space
  Opt-2: Data-Space
- Enumerative
  w/ Pruning Strategy
  w/ ML Bias
- Stochastic/ ML-Driven

Tackling Synthesis for Pandas

Regular DSLs / Grammars won't really cut it:
Large Search Space!

Specification Modality

- Formal Specs
- Program Sketches
- Traces/ Demonstrations
- I/O Examples
- Natural Language

Search Space

- Sketches and Holes (Partial Programs with Holes)
- DSLs / Grammars
- Anything (Arbitrary Text)

Search Algorithm

- Constraint-Solving, CEGIS
  Commonly Requires Translation to SAT/SMT
- Deduction
  Opt-1: Program-Space
  Opt-2: Data-Space
- Enumerative
  w/ Pruning Strategy
  w/ ML Bias
- Stochastic/ ML-Driven

Tackling Synthesis for Pandas

Regular DSLs / Grammars wo

Large

Specification Modality

Natural Language

Search Space

Sketches and Holes (Partial Programs with Holes)

thing ary Text)

Search Algorithm

Constraint-Solving, CEGIS

Deduction

Enumerative

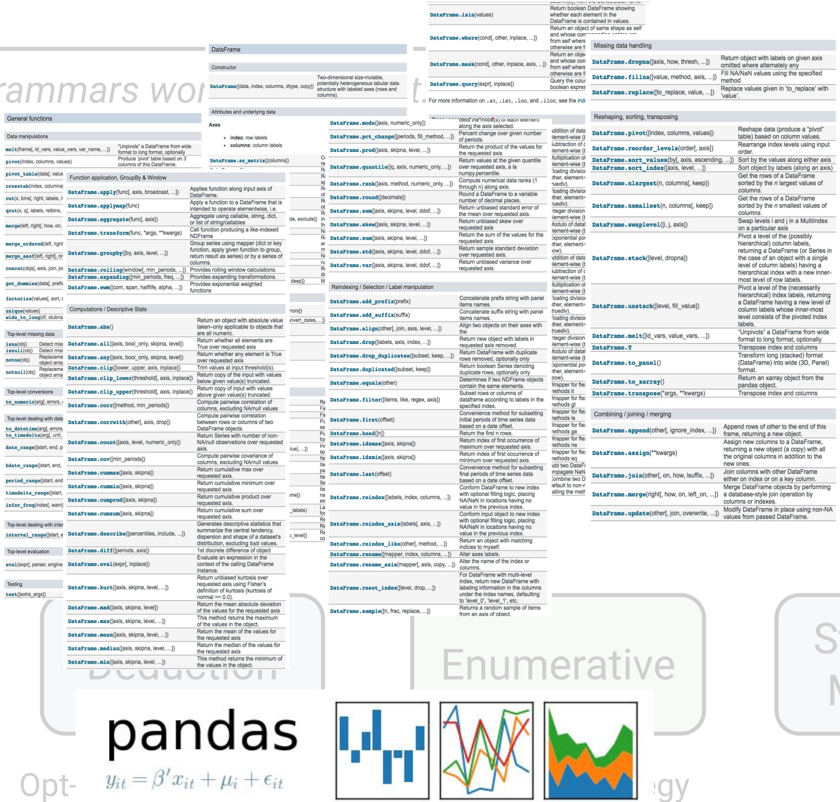
Stochastic/ ML-Driven

Commonly Requires Translation to SAT/SMT

Opt-
Opt-2: Data-Space

gy
w/ ML Bias

> 300 Functions!



Tackling Synthesis for Pandas

Regular DSLs / Grammars won't really cut it:
Search space highly dependent on input (args to Pandas function calls are often constants derived from the table (col-names etc.)

Specification Modality

I/O Examples

Natural Language

Search Space

Sketches and Holes
(Partial Programs with Holes)

DSLs / Grammars

Anything
(Arbitrary Text)

Search Algorithm

Constraint-Solving, CEGIS

Deduction

Enumerative

Stochastic/ ML-Driven

Commonly Requires
Translation to SAT/SMT

Opt-1: Program-Space
Opt-2: Data-Space

w/ Pruning Strategy
w/ ML Bias

Tackling Synthesis for Pandas

Regular DSLs / Grammars won't really cut it:
Search space is hard to define precisely.
Pandas functions have a number of constraints that need to be satisfied for programs to be valid that are difficult to express outside of library 🤔

Specification Modality		I/O Examples	Natural Language	
Search Space	Sketches and Holes (Partial Programs with Holes)	DSLs / Grammars	Anything (Arbitrary Text)	
Search Algorithm	Constraint-Solving, CEGIS	Deduction	Enumerative	Stochastic/ ML-Driven
	Commonly Requires Translation to SAT/SMT	Opt-1: Program-Space Opt-2: Data-Space	w/ Pruning Strategy w/ ML Bias	

Tackling Synthesis for Pandas

Specification Modality

Search Space

Search Algorithm

pandas.DataFrame.drop

```
DataFrame.drop(labels=None, axis=0, index=None, columns=None, level=None, inplace=False, errors='raise')
```

[source]

Drop specified labels from rows or columns.

Remove rows or columns by specifying label names and corresponding axis, or by specifying directly index or column names. When using a multi-index, labels on different levels can be removed by specifying the level. See the *user guide <advanced.shown_levels>* for more information about the now unused levels.

Parameters: **labels** : ***single label or list-like***

Index or column labels to drop.

axis : ***{0 or 'index', 1 or 'columns'}, default 0***

Whether to drop labels from the index (0 or 'index') or columns (1 or 'columns').

`labels` must be a list of index values or columns depending on value of **`axis`**

Commonly Requires Translation to SAT/SMT

Opt-1: Program-Space
Opt-2: Data-Space

w/ Pruning Strategy
w/ ML Bias

Tackling Synthesis for Pandas

We probably need something as close to real-code as possible

Specification Modality

- Formal Specs
- Program Sketches
- Traces/ Demonstrations
- I/O Examples
- Natural Language

Search Space

- Sketches and Holes (Partial Programs with Holes)
- DSLs / Grammars
- Anything (Arbitrary Text)

Search Algorithm

- Constraint-Solving, CEGIS
  Commonly Requires Translation to SAT/SMT
- Deduction
  Opt-1: Program-Space
  Opt-2: Data-Space
- Enumerative
  w/ Pruning Strategy
  w/ ML Bias
- Stochastic/ ML-Driven

Tackling Synthesis for Pandas

We need something a bit more powerful/expressive than DSL/Grammars

Specification Modality

- Formal Specs
- Program Sketches
- Traces/ Demonstrations
- I/O Examples
- Natural Language

??

Search Space

- Sketches and Holes (Partial Programs with Holes)
- DSLs / Grammars
- Anything (Arbitrary Text)

Search Algorithm

- Constraint-Solving, CEGIS
- Deduction
- Enumerative
- Stochastic/ ML-Driven

Commonly Requires Translation to SAT/SMT

Opt-1: Program-Space
Opt-2: Data-Space

w/ Pruning Strategy
w/ ML Bias

Tackling Synthesis for Pandas

Can’t use SAT/SMT modeling for tables

Specification Modality

Formal Specs | Program Sketches | Traces/ Demonstrations | I/O Examples | Natural Language

??

Search Space

Sketches and Holes (Partial Programs with Holes) | DSLs / Grammars | Anything (Arbitrary Text)

Search Algorithm

~~Constraint-Solving, CEGIS~~ | Deduction | Enumerative | Stochastic/ ML-Driven

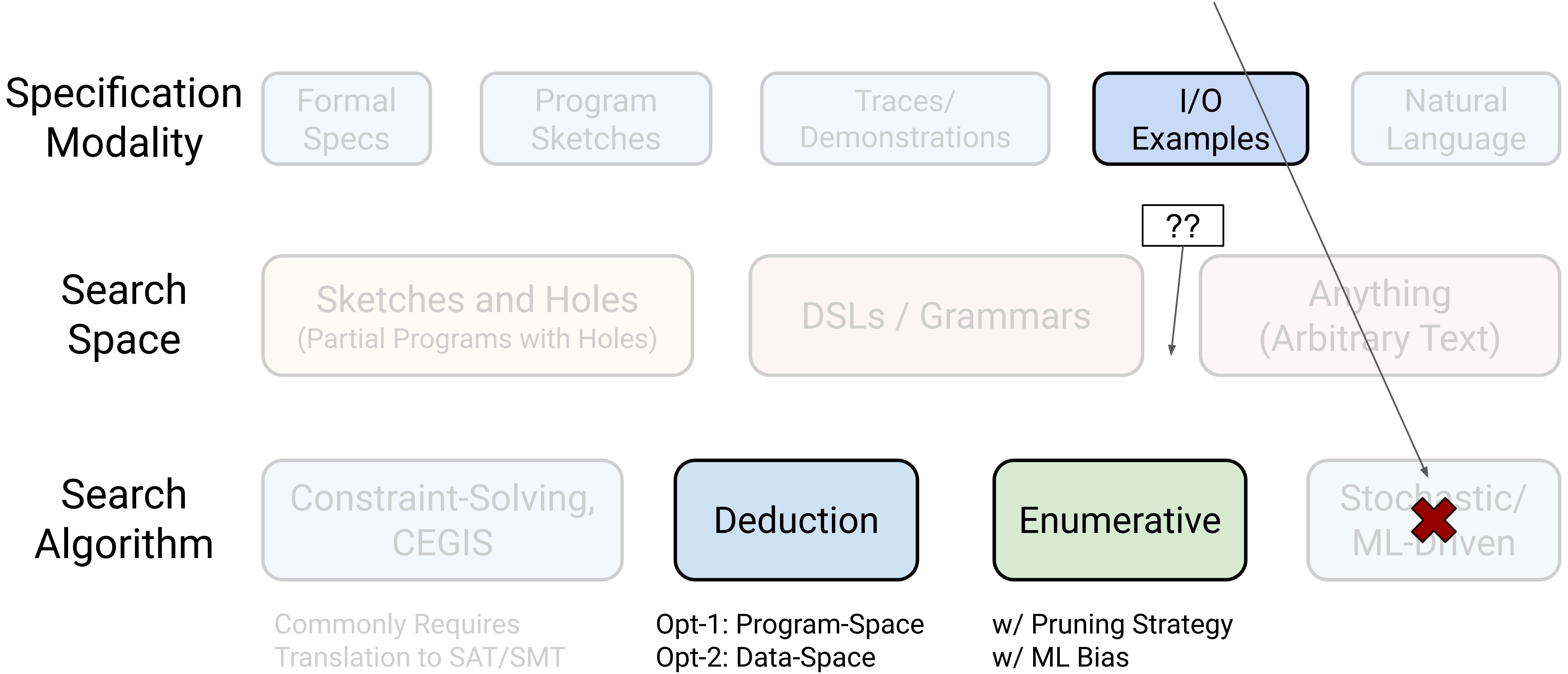
Commonly Requires
Translation to SAT/SMT

Opt-1: Program-Space
Opt-2: Data-Space

w/ Pruning Strategy
w/ ML Bias

Tackling Synthesis for Pandas

Can't do a RobustFill-like approach with a dynamic search space



Tackling Synthesis for Pandas

Difficult to manually write deduction logic for such a large language/API. Requires expertise.

Specification Modality: Formal Specs | Program Sketches | Traces/ Demonstrations | **I/O Examples** | Natural Language

??

Search Space: Sketches and Holes (Partial Programs with Holes) | DSLs / Grammars | Anything (Arbitrary Text)

Search Algorithm: Constraint-Solving, CEGIS | ~~Deduction~~ | **Enumerative** | Stochastic/ ML-Driven

- Constraint-Solving, CEGIS: Commonly Requires Translation to SAT/SMT
- Deduction: Opt-1: Program-Space; Opt-2: Data-Space
- Enumerative: w/ Pruning Strategy; w/ ML Bias

Tackling Synthesis for Pandas

Same, difficult to implement for a large growing/changing API.

Specification Modality	Formal Specs	Program Sketches	Traces/ Demonstrations	I/O Examples	Natural Language

??

Search Space	Sketches and Holes (Partial Programs with Holes)	DSLs / Grammars	Anything (Arbitrary Text)

Search Algorithm	Constraint-Solving, CEGIS	Deduction	Enumerative	Stochastic/ ML-Driven
	Commonly Requires Translation to SAT/SMT	Opt-1: Program-Space Opt-2: Data-Space	w/ Pruning Strategy ✖ w/ ML Bias	

Tackling Synthesis for Pandas

We used the concept of generators from the testing community and added ML bias:
Neural-Backed Generators

Specification Modality

Formal Specs

Program Sketches

Traces/ Demonstrations

I/O Examples

Natural Language

Search Space

Sketches and Holes
(Partial Programs with Holes)

DSLs / Gram…

Generators

…Anything
(…bitrary Text)

Search Algorithm

Constraint-Solving, CEGIS

Deduction

Enumerative

Stochastic/ ML-Driven

Commonly Requires Translation to SAT/SMT

Opt-1: Program-Space
Opt-2: Data-Space

w/ Pruning Strategy
w/ ML Bias

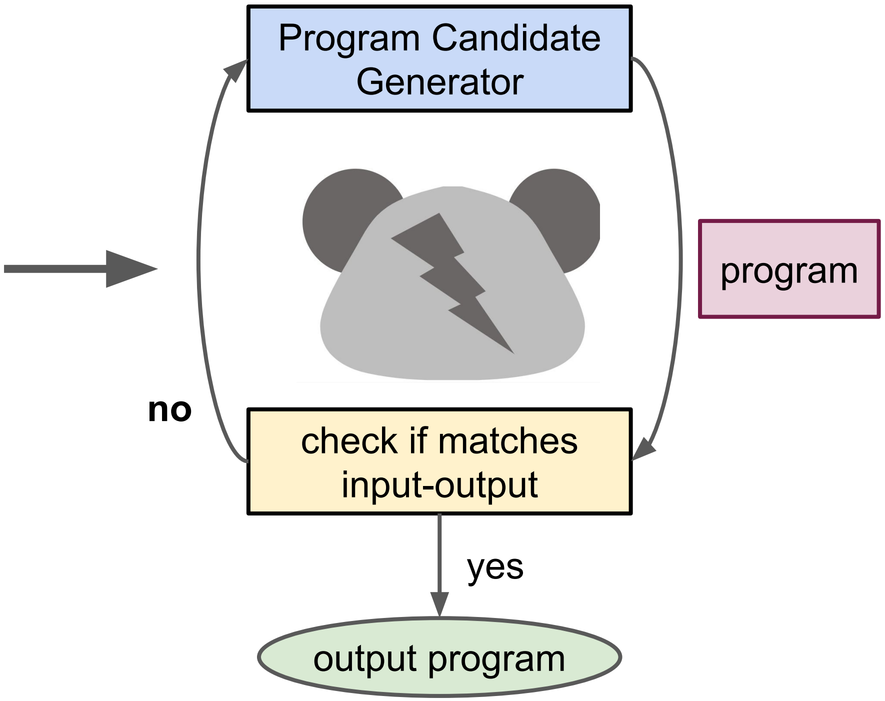
Our Approach : Enumerative Search

input:

	weight	
	kg	lbs
cat	1	2
dog	2	4

output:

		weight
cat	kg	1
	lbs	2
dog	kg	2
	lbs	4



2. Generating Program Candidates

What kind of Programs do we Generate?

Program Candidate Generator

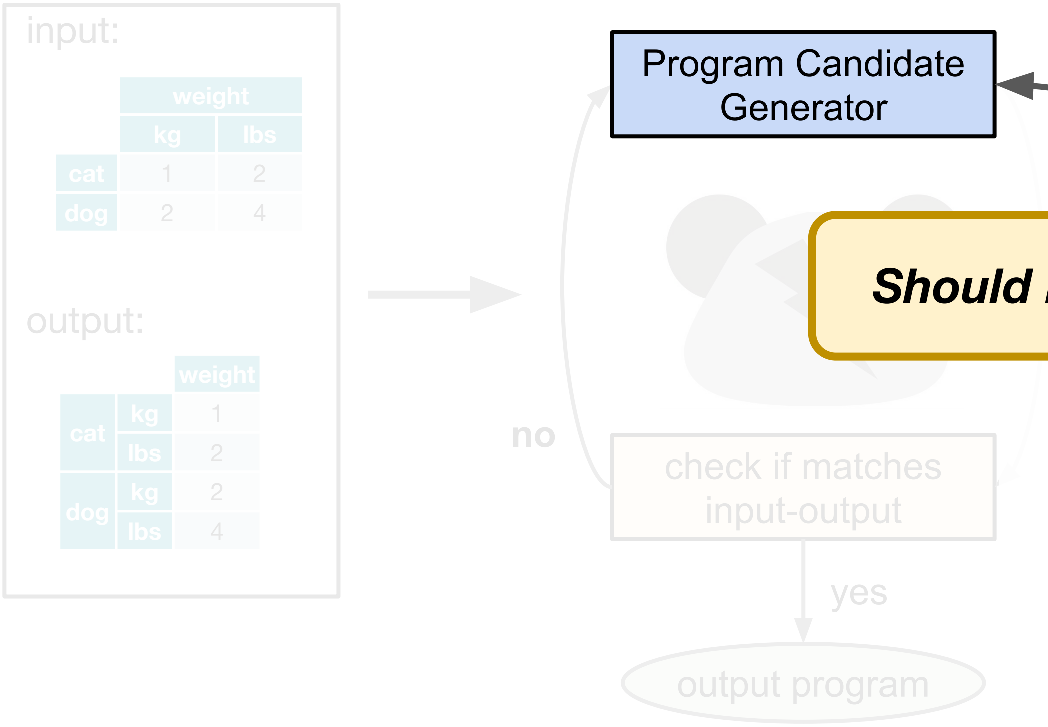
Straight-Line Programs

```
output = input.stack(
              level=[1],
              dropna=True
         )
```

Sequence of Function Calls

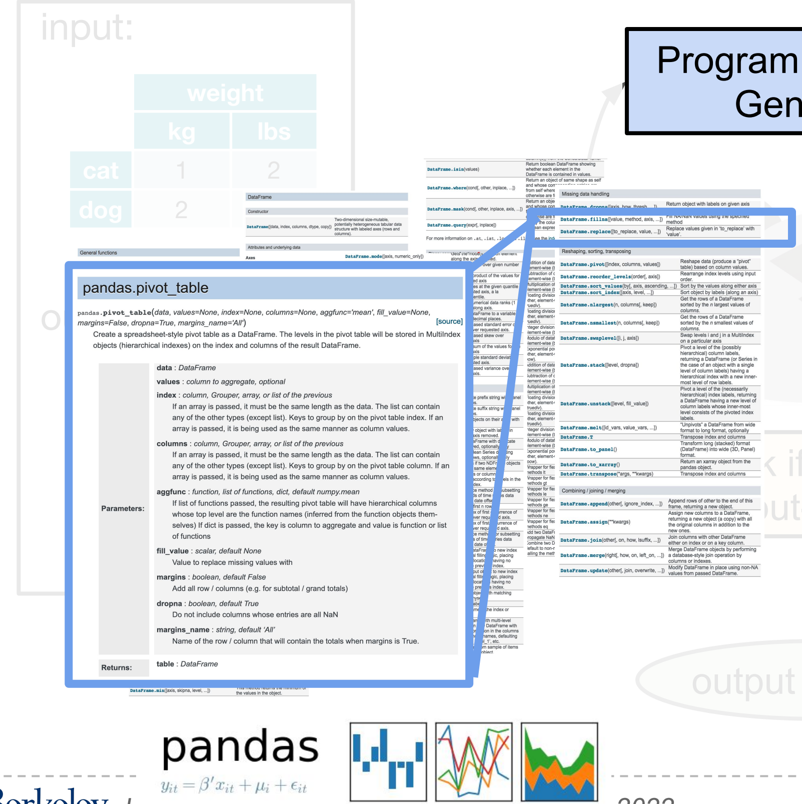
```
v0 = input.melt()
v1 = input.pivot(index = “kg”)
output = v1.reindex()
```

How to Generate Valid Candidates?



Should not return invalid programs

How to Generate Valid Candidates?

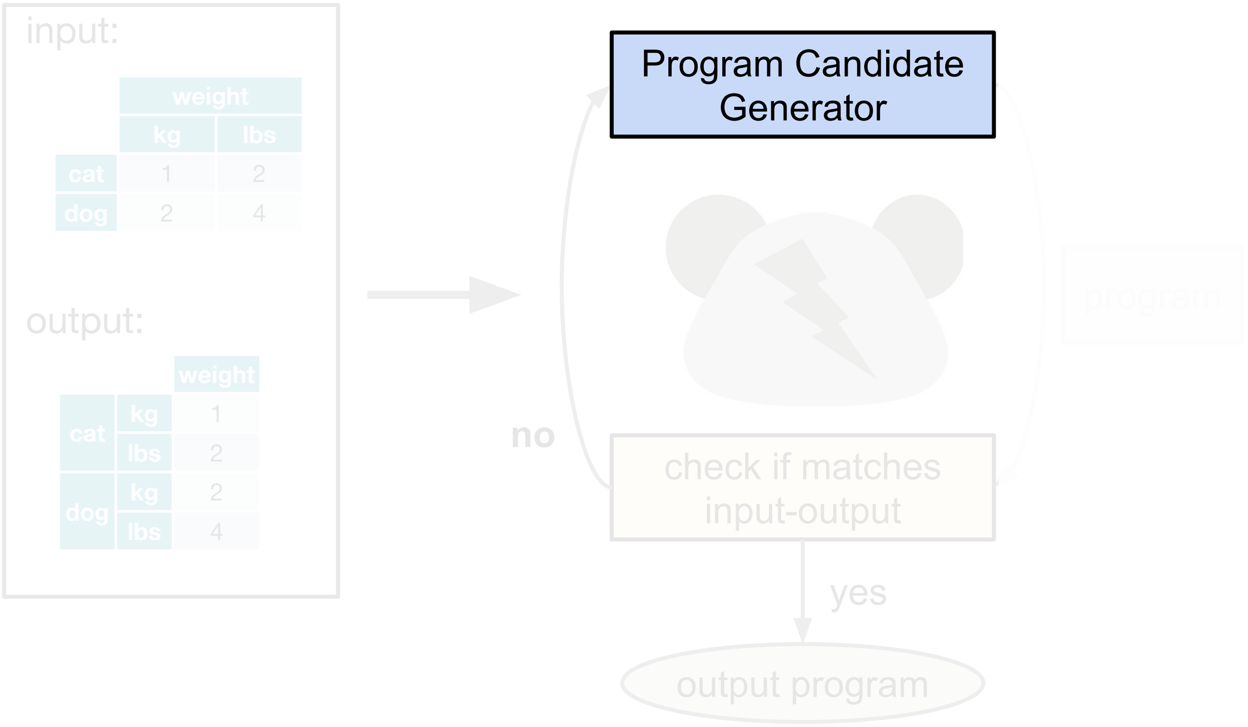


Program Candidate Generator

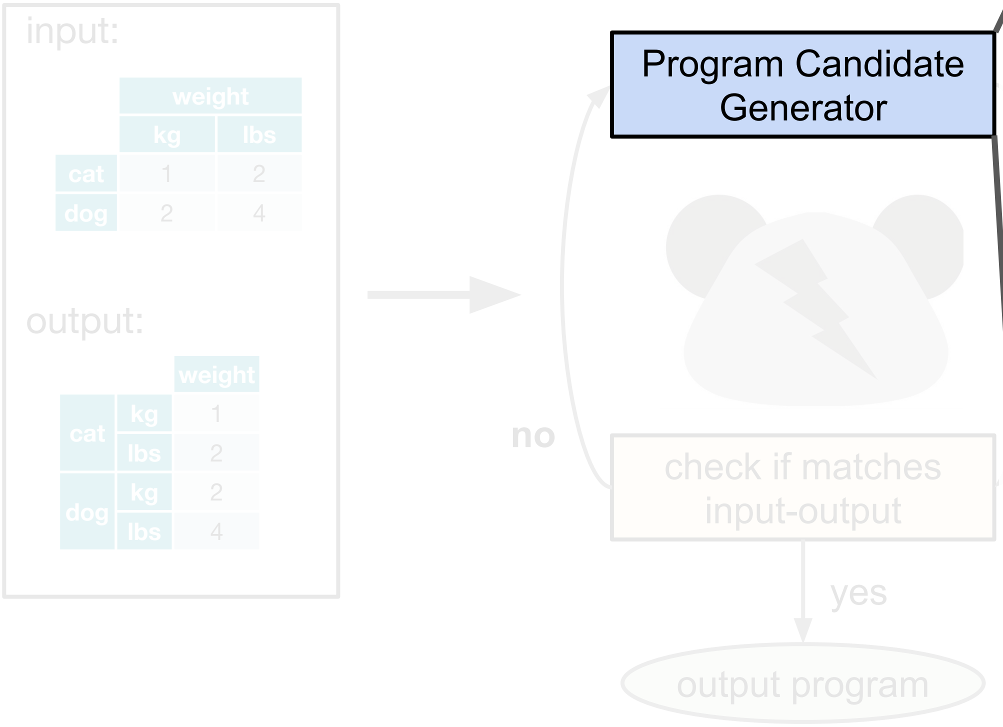
Should not return invalid programs

Need to Capture Constraints

Key Idea #1: Retain Domain Knowledge in a Generator

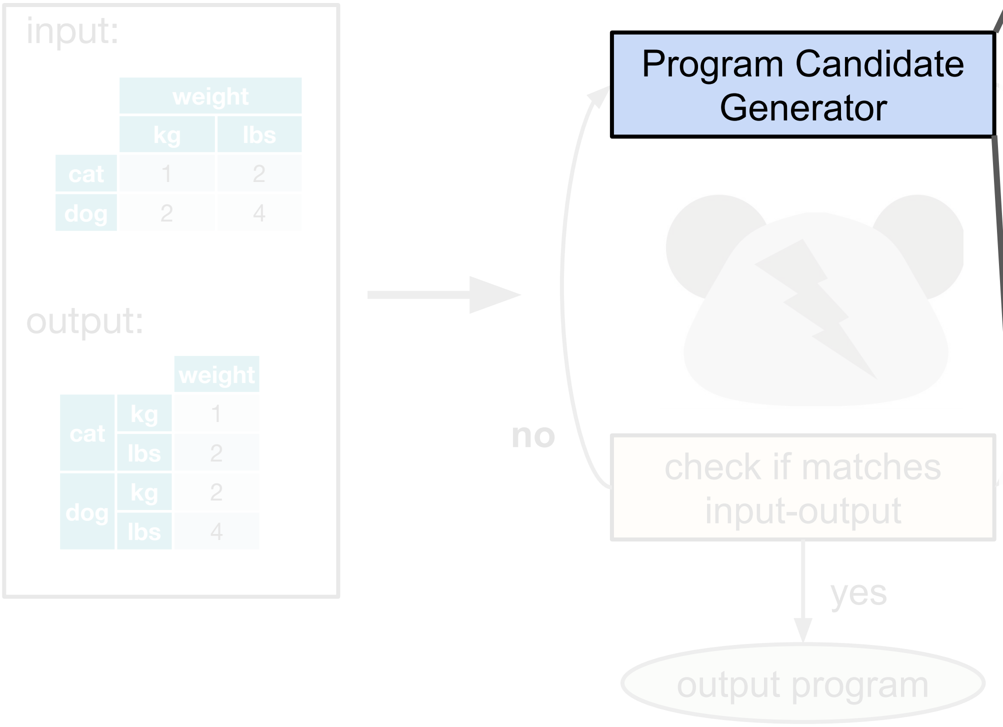


Key Idea #1: Retain Domain Knowledge in a Generator




```
def generate_program(inputs, output):
  fn_seq = Sequence([pivot, melt, drop...])

  for fn in fn_seq:
    if fn == pivot:
      df = Select(inputs)
      arg_col = Select(df.columns)
      arg_idx = Select(df.columns -
                       {arg_col})
      fn.add_args(df, arg_col, arg_idx)

    elif fn == drop:
       df = Select(inputs)
       arg_ax = Select({0,1})
       arg_lbl = Subset(df.index) if arg_ax
                 else Subset(df.columns)
       fn.add_args(df, arg_ax, arg_lbl)

    elif ... :
       # <omitted code...>
    inputs.append(fn.run())
  return fn_seq
```

Key Idea #1: Retain Domain Knowledge in a Generator




```
def generate_program(inputs, output):
  fn_seq = Sequence([pivot, melt, drop...])

  for fn in fn_seq:
    if fn == pivot:
      df = Select(inputs)
      arg_col = Select(df.columns)
      arg_idx = Select(df.columns -
                       {arg_col})
      fn.add_args(df, arg_col, arg_idx)

    elif fn == drop:
      df = Select(inputs)
      arg_ax = Select({0,1})
      arg_lbl = Subset(df.index) if arg_ax
                else Subset(df.columns)
      fn.add_args(df, arg_ax, arg_lbl)

    elif ... :
      # <omitted code...>
    inputs.append(fn.run())
  return fn_seq
```

Key Idea #1: Retain Domain Knowledge in a Generator

input:

	weight	
	kg	lbs
cat	1	2
dog	2	4

outp

Program Candidate Generator

```
def generate_program(inputs, output):
  fn_seq = Sequence([pivot, melt, drop...])

  for fn in fn_seq:
    if fn == pivot:
      df = Select(inputs)
      arg_col = Select(df.columns)
      arg_idx = Select(df.columns -
                       {arg_col})
      fn.add_args(df, arg_col, arg_idx)

    elif fn == drop:
      df = Select(inputs)
      arg_ax = Select({0,1})
      arg_lbl = Subset(df.index) if arg_ax
                else Subset(df.columns)
      fn.add_args(df, arg_ax, arg_lbl)

    elif ... :
      # <omitted code...>
    inputs.append(fn.run())
  return fn_seq
```

The Sequence Operator

Input : List of Elements
Output : Any sequence of elements

Key Idea #1: Retain Domain Knowledge in a Generator

input:

	weight	
	kg	lbs
cat	1	2
dog	2	4

Program Candidate Generator

```
def generate_program(inputs, output):
  fn_seq = Sequence([pivot, melt, drop...])

  for fn in fn_seq:
    if fn == pivot:
      df = Select(inputs)
      arg_col = Select(df.columns)
      arg_idx = Select(df.columns -
                        {arg_col})
      fn.add_args(df, arg_col, arg_idx)

    elif fn == drop:
       df = Select(inputs)
       arg_ax = Select({0,1})
       arg_lbl = Subset(df.index) if arg_ax
                 else Subset(df.columns)
       fn.add_args(df, arg_ax, arg_lbl)

    elif ... :
       # <omitted code...>
    inputs.append(fn.run())
  return fn_seq
```

The Sequence Operator

Input : [pivot, melt, drop]
Output : [pivot]

Key Idea #1: Retain Domain Knowledge in a Generator

Program Candidate Generator

```
def generate_program(inputs, output):
  fn_seq = Sequence([pivot, melt, drop...])

  for fn in fn_seq:
    if fn == pivot:
      df = Select(inputs)
      arg_col = Select(df.columns)
      arg_idx = Select(df.columns -
                       {arg_col})
      fn.add_args(df, arg_col, arg_idx)

    elif fn == drop:
       df = Select(inputs)
       arg_ax = Select({0,1})
       arg_lbl = Subset(df.index) if arg_ax
                 else Subset(df.columns)
       fn.add_args(df, arg_ax, arg_lbl)

    elif ... :
       # <omitted code...>
    inputs.append(fn.run())
  return fn_seq
```

The Sequence Operator

Input : [pivot, melt, drop]
Output : [melt, pivot]

Key Idea #1: Retain Domain Knowledge in a Generator

input:

	weight	
	kg	lbs
cat	1	2
dog	2	4

outp...

Program Candidate Generator

```
def generate_program(inputs, output):
  fn_seq = Sequence([pivot, melt, drop...])

  for fn in fn_seq:
    if fn == pivot:
      df = Select(inputs)
      arg_col = Select(df.columns)
      arg_idx = Select(df.columns -
                       {arg_col})
      fn.add_args(df, arg_col, arg_idx)

    elif fn == drop:
      df = Select(inputs)
      arg_ax = Select({0,1})
      arg_lbl = Subset(df.index) if arg_ax
                else Subset(df.columns)
      fn.add_args(df, arg_ax, arg_lbl)

    elif ... :
      # <omitted code...>
    inputs.append(fn.run())
  return fn_seq
```

The Sequence Operator

Input : [pivot, melt, drop]
Output : [drop, pivot, melt]

Key Idea #1: Retain Domain Knowledge in a Generator

input:

	weight	
	kg	lbs
cat	1	2
dog	2	4

output:

weight

cat

dog

Program Candidate Generator

For each function in the sequence,

```
def generate_program(inputs, output):
  fn_seq = Sequence([pivot, melt, drop...])

  for fn in fn_seq:
    if fn == pivot:
      df = Select(inputs)
      arg_col = Select(df.columns)
      arg_idx = Select(df.columns -
                       {arg_col})
      fn.add_args(df, arg_col, arg_idx)

    elif fn == drop:
       df = Select(inputs)
       arg_ax = Select({0,1})
       arg_lbl = Subset(df.index) if arg_ax
                 else Subset(df.columns)
       fn.add_args(df, arg_ax, arg_lbl)

    elif ... :
       # <omitted code...>
    inputs.append(fn.run())
  return fn_seq
```

Key Idea #1: Retain Domain Knowledge in a Generator

Program Candidate Generator

```
def generate_program(inputs, output):
  fn_seq = Sequence([pivot, melt, drop...])

  for fn in fn_seq:
    if fn == pivot:
      df = Select(inputs)
      arg_col = Select(df.columns)
      arg_idx = Select(df.columns -
                       {arg_col})
      fn.add_args(df, arg_col, arg_idx)

    elif fn == drop:
       df = Select(inputs)
       arg_ax = Select({0,1})
       arg_lbl = Subset(df.index) if arg_ax
                 else Subset(df.columns)
       fn.add_args(df, arg_ax, arg_lbl)

    elif ... :
       # <omitted code...>
    inputs.append(fn.run())
  return fn_seq
```

For each function in the sequence,

Decide the arguments

Key Idea #1: Retain Domain Knowledge in a Generator

input:

	weight	
	kg	lbs
cat	1	2
dog	2	4

output:

weight

cat

dog

Program Candidate Generator

```
def generate_program(inputs, output):
  fn_seq = Sequence([pivot, melt, drop...])

  for fn in fn_seq:
    if fn == pivot:
      df = Select(inputs)
      arg_col = Select(df.columns)
      arg_idx = Select(df.columns -
                       {arg_col})
      fn.add_args(df, arg_col, arg_idx)

    elif fn == drop:
      df = Select(inputs)
      arg_ax = Select({0,1})
      arg_lbl = Subset(df.index) if arg_ax
                else Subset(df.columns)
      fn.add_args(df, arg_ax, arg_lbl)

    elif ... :
      # <omitted code...>
    inputs.append(fn.run())
  return fn_seq
```

For each function in the sequence,

Decide the arguments

On a per-function basis

Key Idea #1: Retain Domain Knowledge in a Generator

input:

	weight	
	kg	lbs
cat	1	2
dog	2	4

output:

	weight
cat	
dog	

Program Candidate Generator

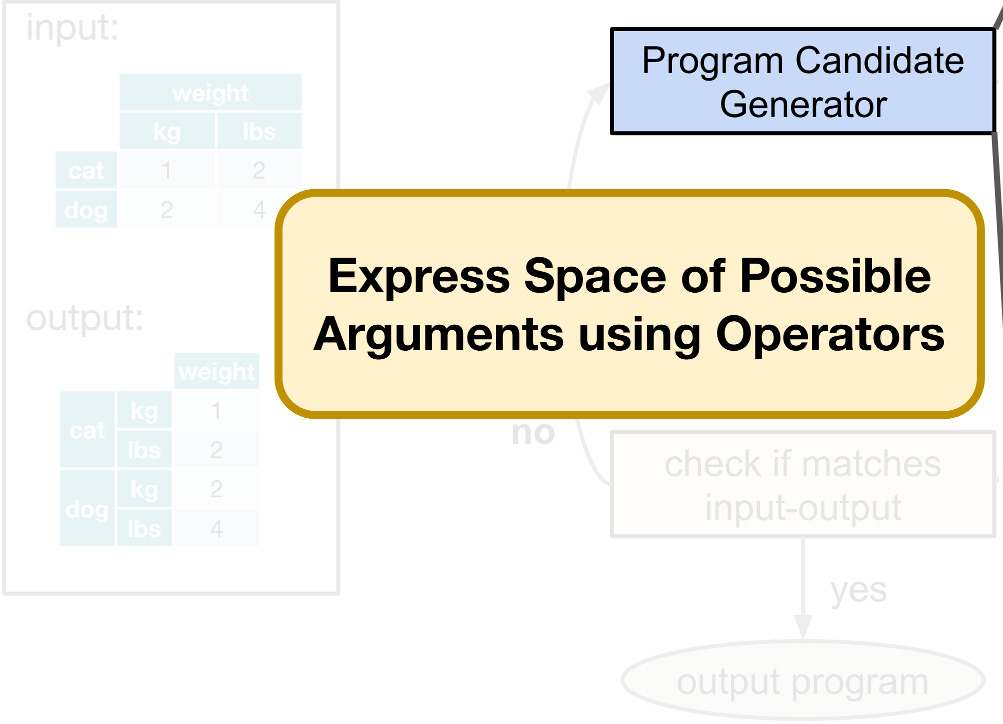
For each function in the sequence,

Decide the arguments

On a per-function basis

```
def generate_program(inputs, output):
  fn_seq = Sequence([pivot, melt, drop...])

  for fn in fn_seq:
    if fn == pivot:
      df = Select(inputs)
      arg_col = Select(df.columns)
      arg_idx = Select(df.columns -
                       {arg_col})
      fn.add_args(df, arg_col, arg_idx)

    elif fn == drop:
      df = Select(inputs)
      arg_ax = Select({0,1})
      arg_lbl = Subset(df.index) if arg_ax
                else Subset(df.columns)
      fn.add_args(df, arg_ax, arg_lbl)

    elif ... :
      # <omitted code...>
    inputs.append(fn.run())
  return fn_seq
```

Key Idea #1: Retain Domain Knowledge in a Generator



Express Space of Possible Arguments using Operators

```
def generate_program(inputs, output):
  fn_seq = Sequence([pivot, melt, drop...])

  for fn in fn_seq:
    if fn == pivot:
      df = Select(inputs)
      arg_col = Select(df.columns)
      arg_idx = Select(df.columns -
                       {arg_col})
      fn.add_args(df, arg_col, arg_idx)

    elif fn == drop:
      df = Select(inputs)
      arg_ax = Select({0,1})
      arg_lbl = Subset(df.index) if arg_ax
                else Subset(df.columns)
      fn.add_args(df, arg_ax, arg_lbl)

    elif ... :
      # <omitted code...>
    inputs.append(fn.run())
  return fn_seq
```

Key Idea #1: Retain Domain Knowledge in a Generator

input:

	weight	
	kg	lbs
cat	1	2
dog	2	4

outp

Program Candidate Generator

```
def generate_program(inputs, output):
  fn_seq = Sequence([pivot, melt, drop...])

  for fn in fn_seq:
    if fn == pivot:
      df = Select(inputs)
      arg_col = Select(df.columns)
      arg_idx = Select(df.columns -
                       {arg_col})
      fn.add_args(df, arg_col, arg_idx)

    elif fn == drop:
      df = Select(inputs)
      arg_ax = Select({0,1})
      arg_lbl = Subset(df.index) if arg_ax
                else Subset(df.columns)
      fn.add_args(df, arg_ax, arg_lbl)

    elif ... :
      # <omitted code...>
    inputs.append(fn.run())
  return fn_seq
```

The Select Operator

Input : List of Elements
Output : Any Element in List

Key Idea #1: Retain Domain Knowledge in a Generator

df

	Date	Category	Expense
0	2018-02-18	Social	98.34
1	2018-02-18	Lunch	245.63
2	2018-02-20	Social	121.89
3	2018-02-20	Lunch	248

`Select(df.columns)`

Input : `['Date', 'Category', 'Expense']`
Output : `'Category'`

```
def generate_program(inputs, output):
  fn_seq = Sequence([pivot, melt, drop...])

  for fn in fn_seq:
    if fn == pivot:
      df = Select(inputs)
      arg_col = Select(df.columns)
      arg_idx = Select(df.columns -
                       {arg_col})
      fn.add_args(df, arg_col, arg_idx)

    elif fn == drop:
      df = Select(inputs)
      arg_ax = Select({0,1})
      arg_lbl = Subset(df.index) if arg_ax
                else Subset(df.columns)
      fn.add_args(df, arg_ax, arg_lbl)

    elif ... :
      # <omitted code...>
    inputs.append(fn.run())
  return fn_seq
```

Key Idea #1: Retain Domain Knowledge in a Generator

input

df

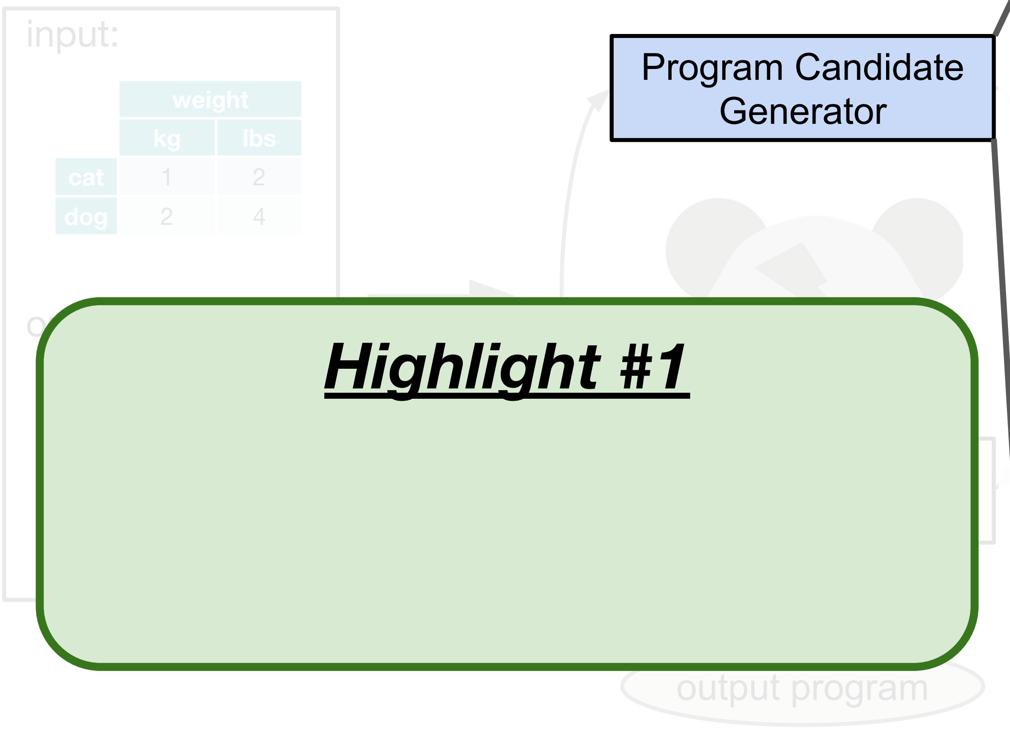
	Date	Category	Expense
0	2018-02-18	Social	98.34
1	2018-02-18	Lunch	245.63
2	2018-02-20	Social	121.89
3	2018-02-20	Lunch	248

`Select(df.columns)`

Input : `[‘Date’, ‘Category’, ‘Expense’]`
Output : `‘Expense’`

```
def generate_program(inputs, output):
  fn_seq = Sequence([pivot, melt, drop...])

  for fn in fn_seq:
    if fn == pivot:
      df = Select(inputs)
      arg_col = Select(df.columns)
      arg_idx = Select(df.columns -
                       {arg_col})
      fn.add_args(df, arg_col, arg_idx)

    elif fn == drop:
      df = Select(inputs)
      arg_ax = Select({0,1})
      arg_lbl = Subset(df.index) if arg_ax
                else Subset(df.columns)
      fn.add_args(df, arg_ax, arg_lbl)

    elif ... :
      # <omitted code...>
    inputs.append(fn.run())
  return fn_seq
```

Key Idea #1: Retain Domain Knowledge in a Generator



<u>Highlight #1</u>

```
def generate_program(inputs, output):
  fn_seq = Sequence([pivot, melt, drop...])

  for fn in fn_seq:
    if fn == pivot:
      df = Select(inputs)
      arg_col = Select(df.columns)
      arg_idx = Select(df.columns -
                       {arg_col})
      fn.add_args(df, arg_col, arg_idx)

    elif fn == drop:
       df = Select(inputs)
       arg_ax = Select({0,1})
       arg_lbl = Subset(df.index) if arg_ax
                 else Subset(df.columns)
       fn.add_args(df, arg_ax, arg_lbl)

    elif ... :
       # <omitted code...>
    inputs.append(fn.run())
  return fn_seq
```

Key Idea #1: Retain Domain Knowledge in a Generator

Program Candidate Generator

```
def generate_program(inputs, output):
  fn_seq = Sequence([pivot, melt, drop...])

  for fn in fn_seq:
    if fn == pivot:
      df = Select(inputs)
      arg_col = Select(df.columns)
      arg_idx = Select(df.columns -
                       {arg_col})
      fn.add_args(df, arg_col, arg_idx)

    elif fn == drop:
       df = Select(inputs)
       arg_ax = Select({0,1})
       arg_lbl = Subset(df.index) if arg_ax
                 else Subset(df.columns)
       fn.add_args(df, arg_ax, arg_lbl)

    elif ... :
       # <omitted code...>
    inputs.append(fn.run())
  return fn_seq
```

Highlight #1

Capture Constraints using Arbitrary Python Code

Key Idea #1: Retain Domain Knowledge in a Generator

input:

	weight	
	kg	lbs
cat	1	2
dog	2	4

Program Candidate Generator

output program

```
def generate_program(inputs, output):
  fn_s                                                  ])
      index != columns
      arg_col = Select(df.columns)
      arg_idx = Select(df.columns -
                           {arg_col})
      fn.add_args(df, arg_col, arg_idx)

    elif fn == drop:
      df = Select(inputs)
      arg_ax = Select({0,1})
      arg_lbl = Subset(df.index) if arg_ax
                  else Subset(df.columns)
      fn.add_args(df, arg_ax, arg_lbl)

    elif ... :
      # <omitted code...>
    inputs.append(fn.run())
  return fn_seq
```

Highlight #1

Capture Constraints using Arbitrary Python Code

Key Idea #1: Retain Domain Knowledge in a Generator

Program Candidate Generator

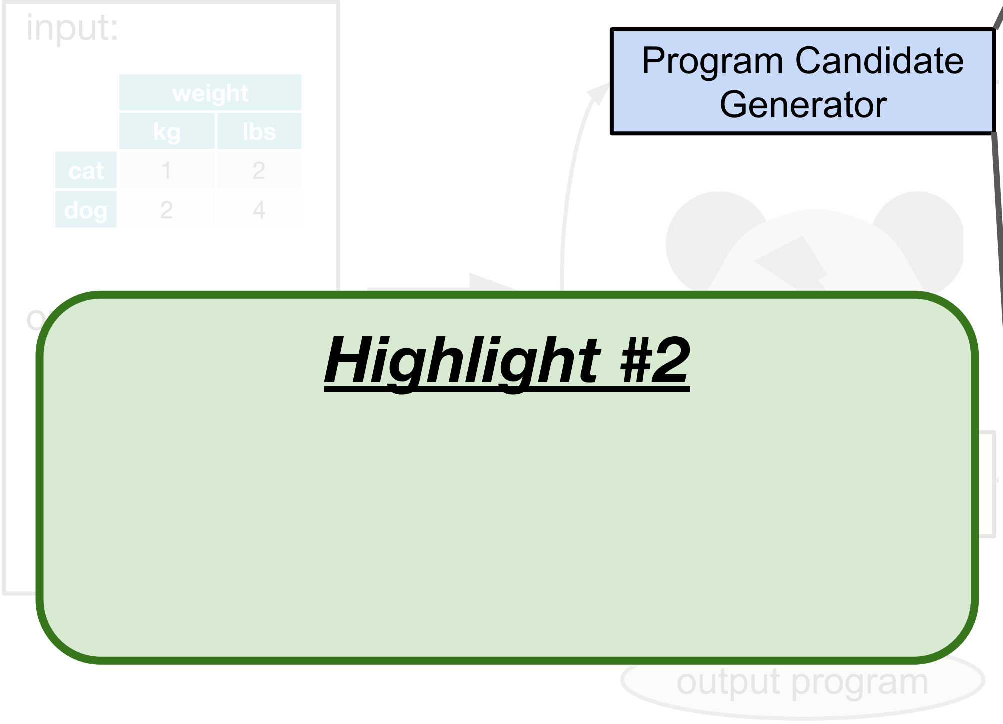
```
def generate_program(inputs, output):
  fn_seq = Sequence([pivot, melt, drop...])

  for fn in fn_seq:
    if fn == pivot:
      df = Select(inputs)
      arg_col = Select(df.columns)

       df = Select(inputs)
       arg_ax = Select({0,1})
       arg_lbl = Subset(df.index) if arg_ax
                 else Subset(df.columns)
       fn.add_args(df, arg_ax, arg_lbl)

    elif ... :
      # <omitted code...>
    inputs.append(fn.run())
  return fn_seq
```

labels consistent with **axis**

Highlight #1

Capture Constraints using Arbitrary Python Code

Key Idea #1: Retain Domain Knowledge in a Generator



<u>Highlight #2</u>

```
def generate_program(inputs, output):
  fn_seq = Sequence([pivot, melt, drop...])

  for fn in fn_seq:
    if fn == pivot:
      df = Select(inputs)
      arg_col = Select(df.columns)
      arg_idx = Select(df.columns -
                        {arg_col})
      fn.add_args(df, arg_col, arg_idx)

    elif fn == drop:
       df = Select(inputs)
       arg_ax = Select({0,1})
       arg_lbl = Subset(df.index) if arg_ax
                 else Subset(df.columns)
       fn.add_args(df, arg_ax, arg_lbl)

    elif ... :
       # <omitted code...>
    inputs.append(fn.run())
  return fn_seq
```

Key Idea #1: Retain Domain Knowledge in a Generator

input:

	weight	
	kg	lbs
cat	1	2
dog	2	4

Program Candidate Generator

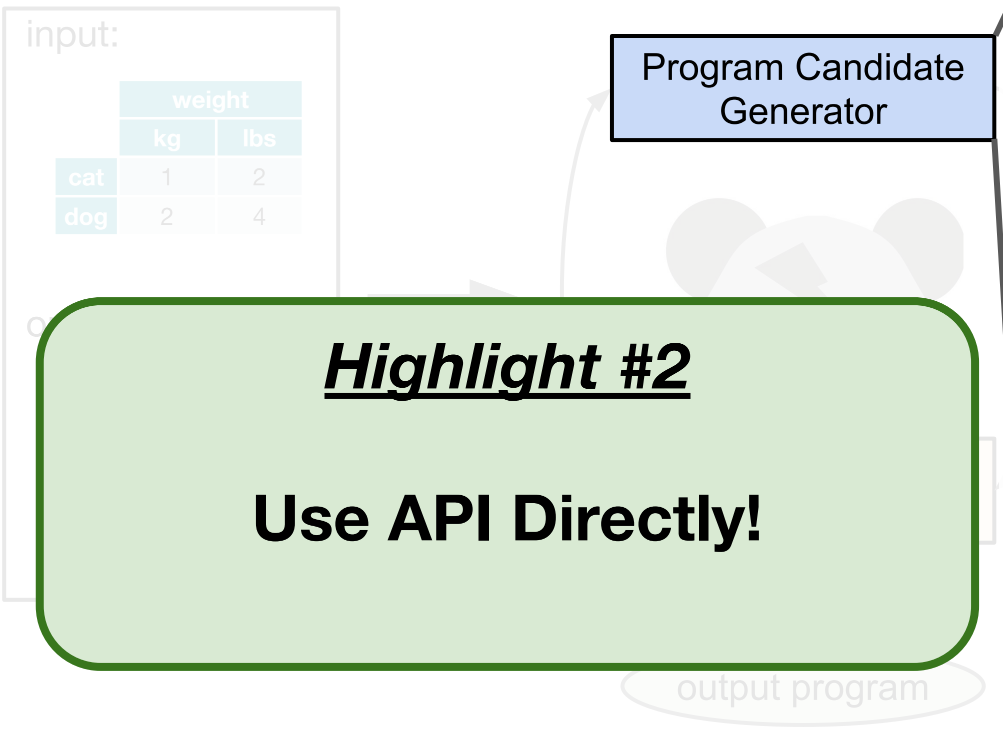
```
def generate_program(inputs, output):
  fn_seq = Sequence([pivot, melt, drop...])

  for fn in fn_seq:
    if fn == pivot:
      df = Select(inputs)
      arg_col = Select(df.columns)
      arg_idx = Select(df.columns -
                       {arg_col})
      fn.add_args(df, arg_col, arg_idx)

    elif fn == drop:
      df = Select(inputs)
      arg_ax = Select({0,1})
      arg_lbl = Subset(df.index) if arg_ax
                else Subset(df.columns)
      fn.add_args(df, arg_ax, arg_lbl)

    elif ... :
      # <omitted code...>
    inputs.append(fn.run())
  return fn_seq
```

<u>Highlight #2</u>

Use API Directly!

output program

Key Idea #1: Retain Domain Knowledge in a Generator



Program Candidate Generator

```
def generate_program(inputs, output):
  fn_seq = Sequence([pivot, melt, drop...])

  for fn in fn_seq:
    if fn == pivot:
      df = Select(inputs)
      arg_col = Select(df.columns)
      arg_idx = Select(df.columns -
                  {arg_col})
      fn.add_args(df, arg_col, arg_idx)

    elif fn == drop:
      df = Select(inputs)
      arg_ax = Select({0,1})
      arg_lbl = Subset(df.index) if arg_ax
                else Subset(df.columns)
      fn.add_args(df, arg_ax, arg_lbl)

    elif ... :
      # <omitted code...>
    inputs.append(fn.run())
  return fn_seq
```

<u>Highlight #2</u>

Use API Directly!

Key Idea #1: Retain Domain Knowledge in a Generator

Program Candidate Generator

Highlight #3

```
def generate_program(inputs, output):
  fn_seq = Sequence([pivot, melt, drop...])

  for fn in fn_seq:
    if fn == pivot:
      df = Select(inputs)
      arg_col = Select(df.columns)
      arg_idx = Select(df.columns -
                       {arg_col})
      fn.add_args(df, arg_col, arg_idx)

    elif fn == drop:
       df = Select(inputs)
       arg_ax = Select({0,1})
       arg_lbl = Subset(df.index) if arg_ax
                 else Subset(df.columns)
       fn.add_args(df, arg_ax, arg_lbl)

    elif ... :
       # <omitted code...>
    inputs.append(fn.run())
  return fn_seq
```

Key Idea #1: Retain Domain Knowledge in a Generator

input:

	weight	
	kg	lbs
cat	1	2
dog	2	4

Program Candidate Generator

Highlight #3

Delegate Non-Determinism to Simple Operators

output program

```
def generate_program(inputs, output):
  fn_seq = Sequence([pivot, melt, drop...])

  for fn in fn_seq:
    if fn == pivot:
      df = Select(inputs)
      arg_col = Select(df.columns)
      arg_idx = Select(df.columns -
                       {arg_col})
      fn.add_args(df, arg_col, arg_idx)

    elif fn == drop:
      df = Select(inputs)
      arg_ax = Select({0,1})
      arg_lbl = Subset(df.index) if arg_ax
                else Subset(df.columns)
      fn.add_args(df, arg_ax, arg_lbl)

    elif ... :
      # <omitted code...>
    inputs.append(fn.run())
  return fn_seq
```

Key Idea #1: Retain Domain Knowledge in a Generator

Similar in spirit to
Quick-Check Generators
[Claessen and Hughes ‘00]

<u>Highlight #3</u>

Delegate Non-Determinism to Simple Operators

```
def generate_program(inputs, output):
  fn_seq = Sequence([pivot, melt, drop...])

  for fn in fn_seq:
    if fn == pivot:
      df = Select(inputs)
      arg_col = Select(df.columns)
      arg_idx = Select(df.columns -
                        {arg_col})
      fn.add_args(df, arg_col, arg_idx)

    elif fn == drop:
      df = Select(inputs)
      arg_ax = Select({0,1})
      arg_lbl = Subset(df.index) if arg_ax
                else Subset(df.columns)
      fn.add_args(df, arg_ax, arg_lbl)

    elif ... :
      # <omitted code...>
    inputs.append(fn.run())
  return fn_seq
```

Key Idea #1: Retain Domain Knowledge in a Generator

Operators analogous to **Holes** in Sketch
[Solar-Lezama ‘08]

<u>Highlight #3</u>

Delegate Non-Determinism to Simple Operators

```
def generate_program(inputs, output):
  fn_seq = Sequence([pivot, melt, drop...])

  for fn in fn_seq:
    if fn == pivot:
      df = Select(inputs)
      arg_col = Select(df.columns)
      arg_idx = Select(df.columns -
                       {arg_col})
      fn.add_args(df, arg_col, arg_idx)

    elif fn == drop:
      df = Select(inputs)
      arg_ax = Select({0,1})
      arg_lbl = Subset(df.index) if arg_ax
                else Subset(df.columns)
      fn.add_args(df, arg_ax, arg_lbl)

    elif ... :
      # <omitted code...>
    inputs.append(fn.run())
  return fn_seq
```

Key Idea #1: Retain Domain Knowledge in a Generator

input:

	weight	
	kg	lbs
cat	1	2
dog	2	4

Program Candidate Generator

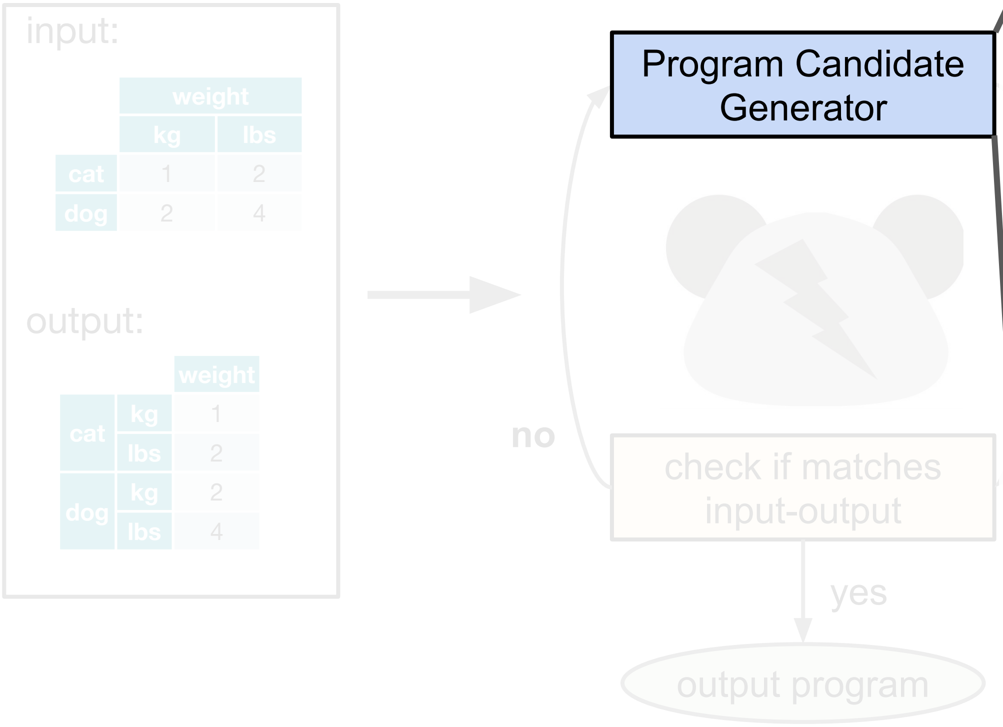
We strongly believe API expertise is sufficient to write Generators

output program

```
def generate_program(inputs, output):
  fn_seq = Sequence([pivot, melt, drop...])

  for fn in fn_seq:
    if fn == pivot:
      df = Select(inputs)
      arg_col = Select(df.columns)
      arg_idx = Select(df.columns -
                       {arg_col})
      fn.add_args(df, arg_col, arg_idx)

    elif fn == drop:
      df = Select(inputs)
      arg_ax = Select({0,1})
      arg_lbl = Subset(df.index) if arg_ax
                else Subset(df.columns)
      fn.add_args(df, arg_ax, arg_lbl)

    elif ... :
      # <omitted code...>
    inputs.append(fn.run())
  return fn_seq
```

3. Controlling Non-Determinism

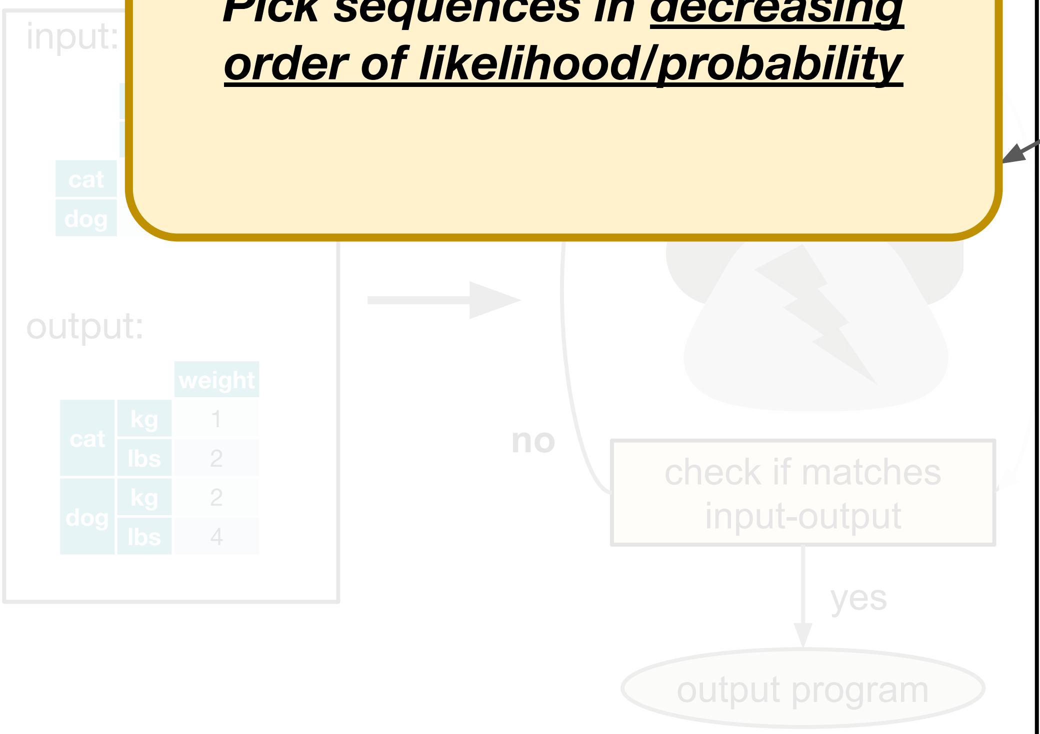
Key Idea #2 : Smartly Control Non-Determinism




```python
def generate_program(inputs, output):
  fn_seq = Sequence([pivot, melt, drop...])

  for fn in fn_seq:
    if fn == pivot:
      df = Select(inputs)
      arg_col = Select(df.columns)
      arg_idx = Select(df.columns -
                       {arg_col})
      fn.add_args(df, arg_col, arg_idx)

    elif fn == drop:
      df = Select(inputs)
      arg_ax = Select({0,1})
      arg_lbl = Subset(df.index) if arg_ax
                else Subset(df.columns)
      fn.add_args(df, arg_ax, arg_lbl)

    elif ... :
      # <omitted code...>
    inputs.append(fn.run())
  return fn_seq
```

Key Idea #2 : Smartly Control Non-Determinism

Pick sequences in <u>decreasing order of likelihood/probability</u>

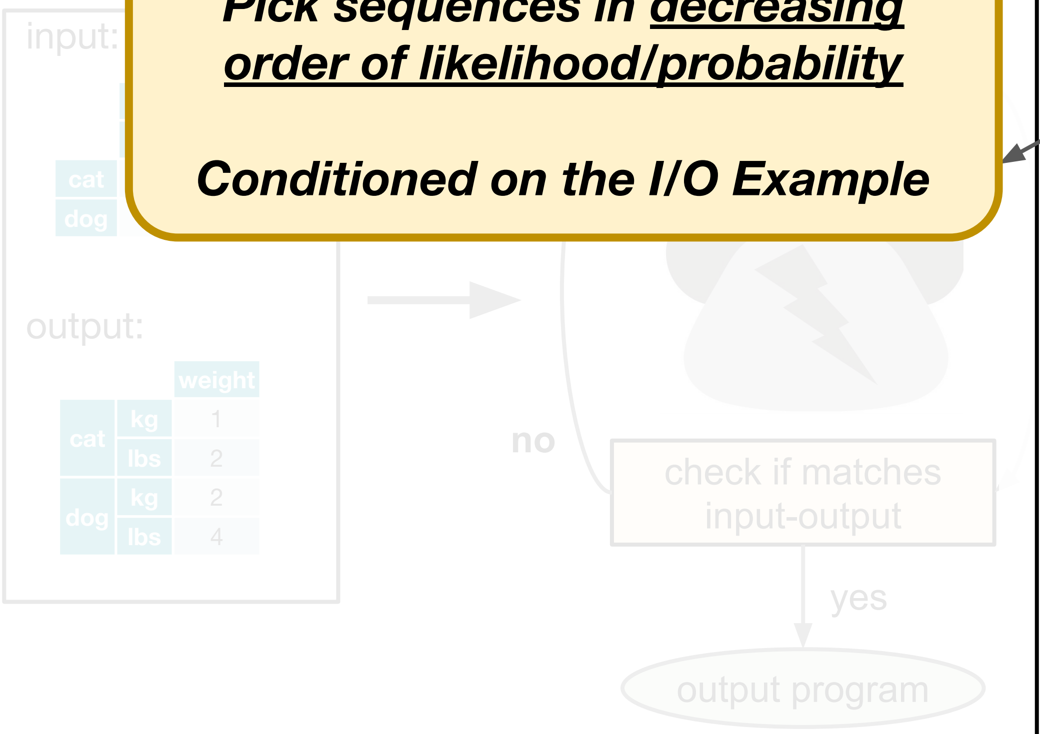



```
def generate_program(inputs, output):
  fn_seq = Sequence([pivot, melt, drop...])

  for fn in fn_seq:
    if fn == pivot:
      df = Select(inputs)
      arg_col = Select(df.columns)
      arg_idx = Select(df.columns -
                    {arg_col})
      fn.add_args(df, arg_col, arg_idx)

    elif fn == drop:
      df = Select(inputs)
      arg_ax = Select({0,1})
      arg_lbl = Subset(df.index) if arg_ax
                else Subset(df.columns)
      fn.add_args(df, arg_ax, arg_lbl)

    elif ... :
      # <omitted code...>
    inputs.append(fn.run())
  return fn_seq
```

Key Idea #2 : Smartly Control Non-Determinism

Pick sequences in <u>decreasing order of likelihood/probability</u>

Conditioned on the I/O Example




```
def generate_program(inputs, output):
  fn_seq = Sequence([pivot, melt, drop...])

  for fn in fn_seq:
    if fn == pivot:
      df = Select(inputs)
      arg_col = Select(df.columns)
      arg_idx = Select(df.columns -
                       {arg_col})
      fn.add_args(df, arg_col, arg_idx)

    elif fn == drop:
      df = Select(inputs)
      arg_ax = Select({0,1})
      arg_lbl = Subset(df.index) if arg_ax
                else Subset(df.columns)
      fn.add_args(df, arg_ax, arg_lbl)

    elif ... :
      # <omitted code...>
    inputs.append(fn.run())
  return fn_seq
```

Key Idea #2 : Smartly Control Non-Determinism

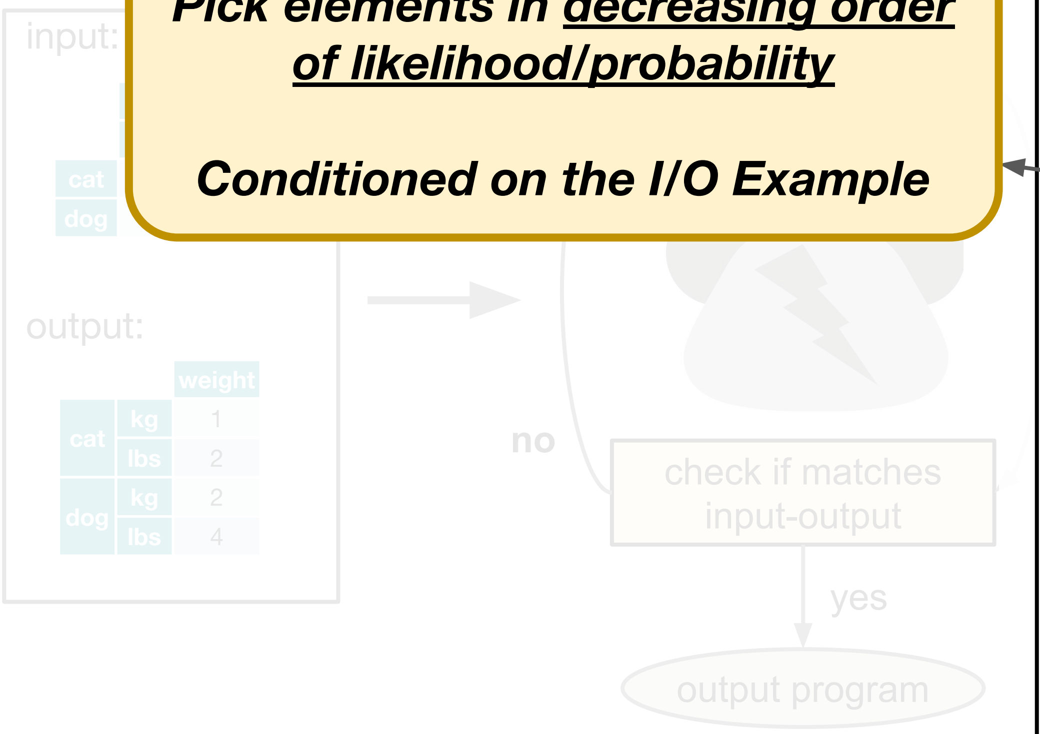
Pick sequences in <u>decreasing order of likelihood/probability</u>

Conditioned on the I/O Example

$\mathbf{P}$(all sequences using [pivot, drop, …] | input, output)

```
def generate_program(inputs, output):
  fn_seq = Sequence([pivot, melt, drop...])

  for fn in fn_seq:
    if fn == pivot:
      df = Select(inputs)
      arg_col = Select(df.columns)
      arg_idx = Select(df.columns -
                       {arg_col})
      fn.add_args(df, arg_col, arg_idx)

    elif fn == drop:
      df = Select(inputs)
      arg_ax = Select({0,1})
      arg_lbl = Subset(df.index) if arg_ax
                else Subset(df.columns)
      fn.add_args(df, arg_ax, arg_lbl)

    elif ... :
      # <omitted code...>
    inputs.append(fn.run())
  return fn_seq
```

Key Idea #2 : Smartly Control Non-Determinism



Pick elements in <u>decreasing order of likelihood/probability</u>

Conditioned on the I/O Example

```
def generate_program(inputs, output):
  fn_seq = Sequence([pivot, melt, drop...])

  for fn in fn_seq:
    if fn == pivot:
      df = Select(inputs)
      arg_col = Select(df.columns)
      arg_idx = Select(df.columns -
                       {arg_col})
      fn.add_args(df, arg_col, arg_idx)

    elif fn == drop:
      df = Select(inputs)
      arg_ax = Select({0,1})
      arg_lbl = Subset(df.index) if arg_ax
                else Subset(df.columns)
      fn.add_args(df, arg_ax, arg_lbl)

    elif ... :
      # <omitted code...>
    inputs.append(fn.run())
  return fn_seq
```

Key Idea #2 : Smartly Control Non-Determinism

Pick elements in <u>decreasing order of likelihood/probability</u>

Conditioned on the I/O Example

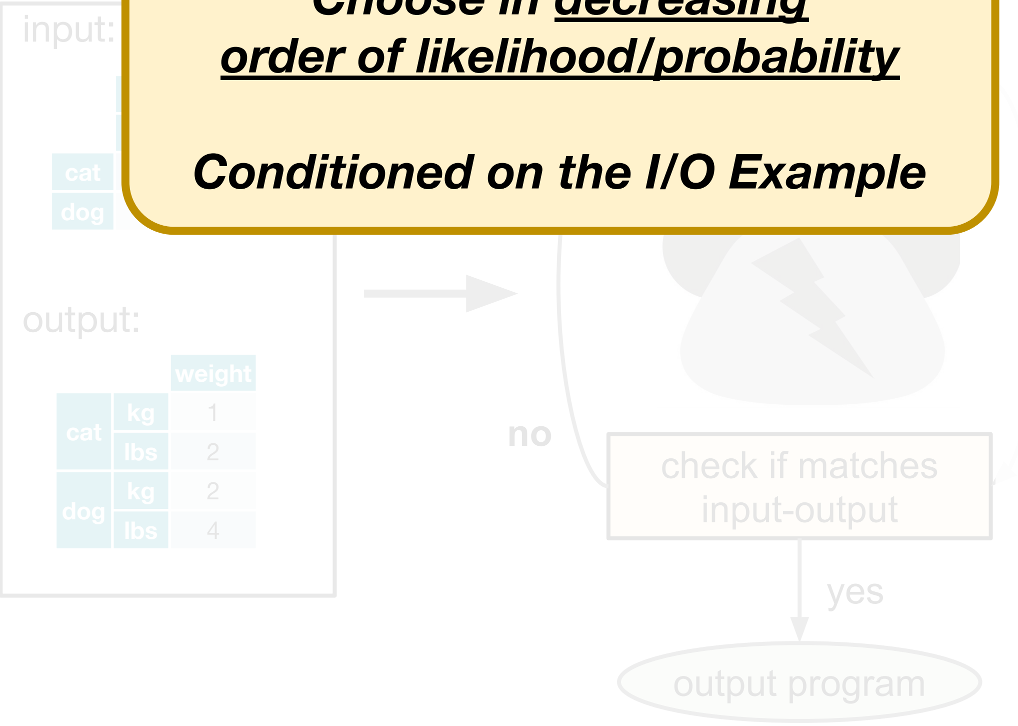
$$\mathbf{P}(\texttt{df.columns} \mid \texttt{input}, \texttt{output})$$

```
def generate_program(inputs, output):
  fn_seq = Sequence([pivot, melt, drop...])

  for fn in fn_seq:
    if fn == pivot:
      df = Select(inputs)
      arg_col = Select(df.columns)
      arg_idx = Select(df.columns -
                       {arg_col})
      fn.add_args(df, arg_col, arg_idx)

    elif fn == drop:
      df = Select(inputs)
      arg_ax = Select({0,1})
      arg_lbl = Subset(df.index) if arg_ax
                else Subset(df.columns)
      fn.add_args(df, arg_ax, arg_lbl)

    elif ... :
      # <omitted code...>
    inputs.append(fn.run())
  return fn_seq
```

Key Idea #2 : Smartly Control Non-Determinism

Choose in <u>decreasing</u>
<u>order of likelihood/probability</u>

Conditioned on the I/O Example

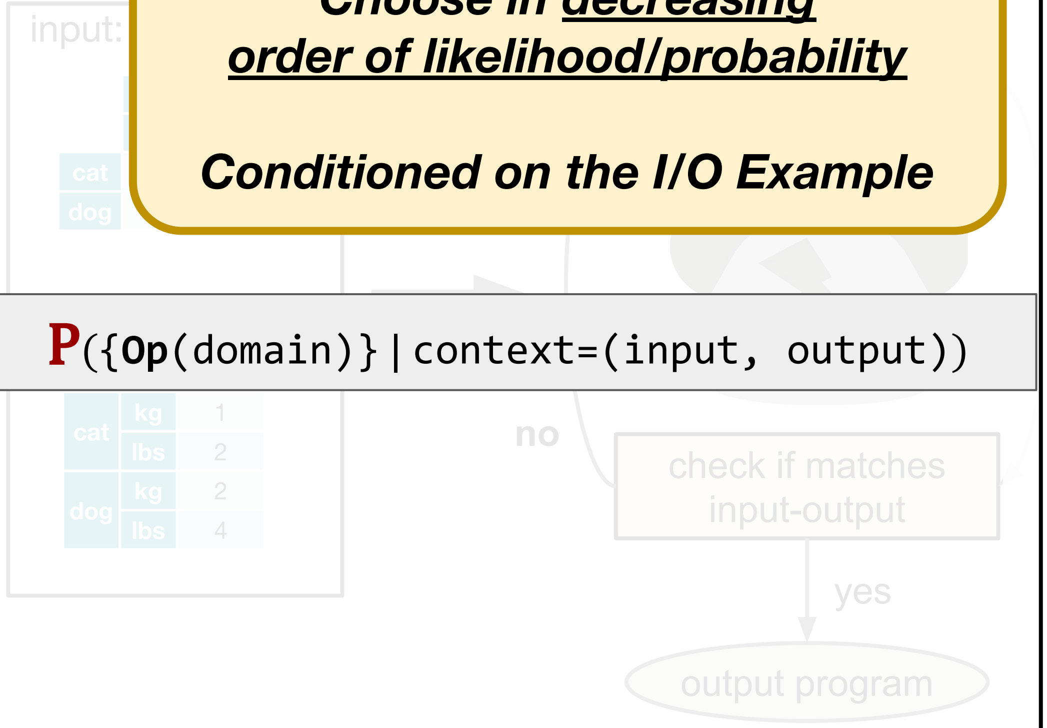



```
def generate_program(inputs, output):
  fn_seq = Sequence([pivot, melt, drop...])

  for fn in fn_seq:
    if fn == pivot:
      df = Select(inputs)
      arg_col = Select(df.columns)
      arg_idx = Select(df.columns -
                       {arg_col})
      fn.add_args(df, arg_col, arg_idx)

    elif fn == drop:
      df = Select(inputs)
      arg_ax = Select({0,1})
      arg_lbl = Subset(df.index) if arg_ax
                else Subset(df.columns)
      fn.add_args(df, arg_ax, arg_lbl)

    elif ... :
      # <omitted code...>
    inputs.append(fn.run())
  return fn_seq
```

Key Idea #2 : Smartly Control Non-Determinism

Choose in <u>decreasing order of likelihood/probability</u>

Conditioned on the I/O Example



$$P(\{\texttt{Op}(\texttt{domain})\} \mid \texttt{context=(input, output)})$$

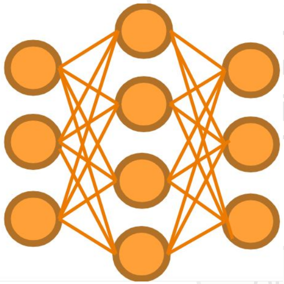
```
def generate_program(inputs, output):
  fn_seq = Sequence([pivot, melt, drop...])

  for fn in fn_seq:
    if fn == pivot:
      df = Select(inputs)
      arg_col = Select(df.columns)
      arg_idx = Select(df.columns -
                       {arg_col})
      fn.add_args(df, arg_col, arg_idx)

    elif fn == drop:
      df = Select(inputs)
      arg_ax = Select({0,1})
      arg_lbl = Subset(df.index) if arg_ax
                else Subset(df.columns)
      fn.add_args(df, arg_ax, arg_lbl)

    elif ... :
      # <omitted code...>
    inputs.append(fn.run())
  return fn_seq
```

Key Idea #2 : Smartly Control Non-Determinism using Neural Models

Choose in <u>decreasing</u>
<u>order of likelihood/probability</u>

Conditioned on the I/O Example

P({**Op**(domain)} | context=(input, output))



```
def generate_program(inputs, output):
  fn_seq = Sequence([pivot, melt, drop...])

  for fn in fn_seq:
    if fn == pivot:
      df = Select(inputs)
      arg_col = Select(df.columns)
      arg_idx = Select(df.columns -
                       {arg_col})
      fn.add_args(df, arg_col, arg_idx)

    elif fn == drop:
      df = Select(inputs)
      arg_ax = Select({0,1})
      arg_lbl = Subset(df.index) if arg_ax
                else Subset(df.columns)
      fn.add_args(df, arg_ax, arg_lbl)

    elif ... :
      # <omitted code...>
    inputs.append(fn.run())
  return fn_seq
```

4. Neural Network Architecture

Neural Architecture using Example

Input

	Date	Category	Expense
0	2018-02-18	Social	98.34
1	2018-02-18	Lunch	245.63
2	2018-02-20	Social	121.89
3	2018-02-20	Lunch	248

Output

	Lunch	Social
2018-02-18	245.63	98.34
2018-02-20	248	121.89

```
fn_seq = Sequence([pivot, melt, drop...])
```

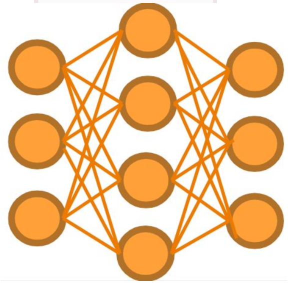
Neural Architecture using Example

Input

	Date	Category	Expense
0	2018-02-18	Social	98.34
1	2018-02-18	Lunch	245.63
2	2018-02-20	Social	121.89
3	2018-02-20	Lunch	248

Output

	Lunch	Social
2018-02-18	245.63	98.34
2018-02-20	248	121.89



Model for **Sequence**

```
fn_seq = Sequence([pivot, melt, drop...])
```

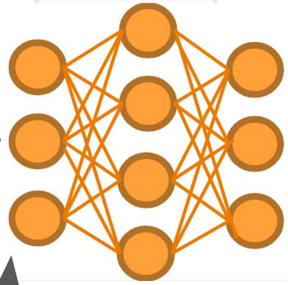
Neural Architecture using Example

Input

	Date	Category	Expense
0	2018-02-18	Social	98.34
1	2018-02-18	Lunch	245.63
2	2018-02-20	Social	121.89
3	2018-02-20	Lunch	248

Output

	Lunch	Social
2018-02-18	245.63	98.34
2018-02-20	248	121.89



Model for **Sequence**

```
fn_seq = Sequence([pivot, melt, drop...])
```

Neural Architecture using Example

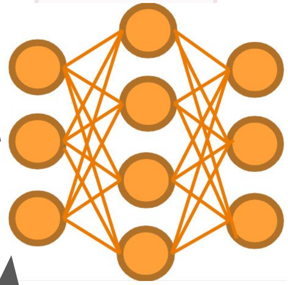
Input

	Date	Category	Expense
0	2018-02-18	Social	98.34
1	2018-02-18	Lunch	245.63
2	2018-02-20	Social	121.89
3	2018-02-20	Lunch	248

Output

	Lunch	Social
2018-02-18	245.63	98.34
2018-02-20	248	121.89

P(all sequences using [pivot, drop, …] | input, output)



Model for **Sequence**

```
fn_seq = Sequence([pivot, melt, drop...])
```

4. (a) How to Encode?

Key Idea #3 : Graph-Based Encoding

Input

	Date	Category	Expense
0	2018-02-18	Social	98.34
1	2018-02-18	Lunch	245.63
2	2018-02-20	Social	121.89
3	2018-02-20	Lunch	248

Output

	Lunch	Social
2018-02-18	245.63	98.34
2018-02-20	248	121.89

Key Insight #1

Relationships between values matter,
Not the values themselves

```
fn_seq = Sequence([pivot, melt, drop...])
```

Key Idea #3 : Graph-Based Encoding

Input

	Date	Category	Expense
0	2018-02-18	Social	98.34
1	2018-02-18	Lunch	245.63
2	2018-02-20	Social	121.89
3	2018-02-20	Lunch	248

Output

	Lunch	Social
2018-02-18	245.63	98.34
2018-02-20	248	121.89

Key Insight #1

Relationships between values matter,
Not the values themselves

```
fn_seq = Sequence([pivot, melt, drop...])
```

Key Idea #3 : Graph-Based Encoding

Input

	Date	Category	Expense
0	2018-02-18	Social	98.34
1	2018-02-18	Lunch	245.63
2	2018-02-20	Social	121.89
3	2018-02-20	**STR**	248

Output **EQUAL**

	STR	Social
2018-02-18	245.63	98.34
2018-02-20	248	121.89

<u>Key Insight #1</u>

Relationships between values matter,
Not the values themselves

```
fn_seq = Sequence([pivot, melt, drop...])
```

Key Idea #3 : Graph-Based Encoding

Input

	Date	Category	Expense
0	2018-02-18	Social	98.34
1	2018-02-18	Lunch	245.63
2	2018-02-20	Social	121.89
3	2018-02-20	**STR**	248

Output **EQUAL**

	STR	Social
2018-02-18	245.63	98.34
2018-02-20	248	121.89

Key Insight #1
Relationships between values matter,
Not the values themselves

Key Insight #2
These relationships can be
encoded as a graph

```
fn_seq = Sequence([pivot, melt, drop...])
```

Key Idea #3 : Graph-Based Encoding

Input

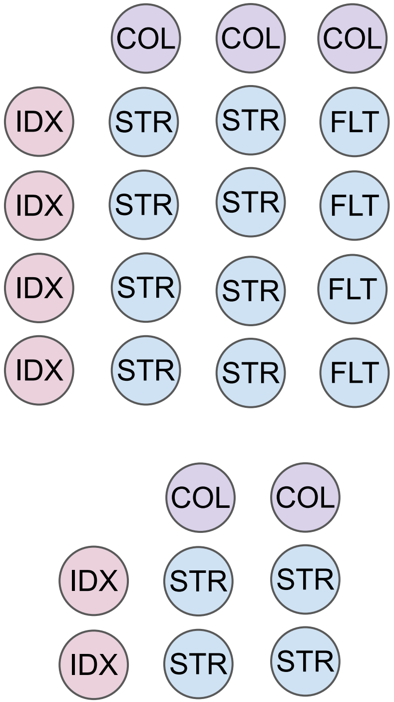
	Date	Category	Expense
0	2018-02-18	Social	98.34
1	2018-02-18	Lunch	245.63
2	2018-02-20	Social	121.89
3	2018-02-20	Lunch	248

Output

	Lunch	Social
2018-02-18	245.63	98.34
2018-02-20	248	121.89

Represent values as nodes

Only retain types




```
fn_seq = Sequence([pivot, melt, drop...])
```

Key Idea #3 : Graph-Based Encoding

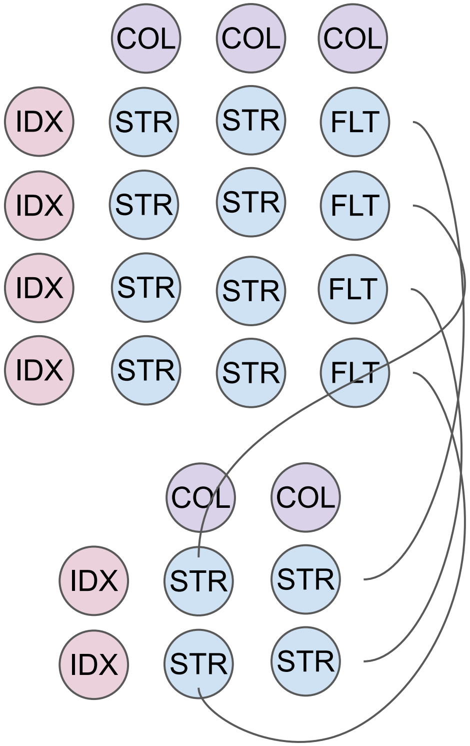
Input

	Date	Category	Expense
0	2018-02-18	Social	98.34
1	2018-02-18	Lunch	245.63
2	2018-02-20	Social	121.89
3	2018-02-20	Lunch	248

Output

	Lunch	Social
2018-02-18	245.63	98.34
2018-02-20	248	121.89

Use *EQUALITY* edges to capture data movement




```
fn_seq = Sequence([pivot, melt, drop...])
```

Key Idea #3 : Graph-Based Encoding

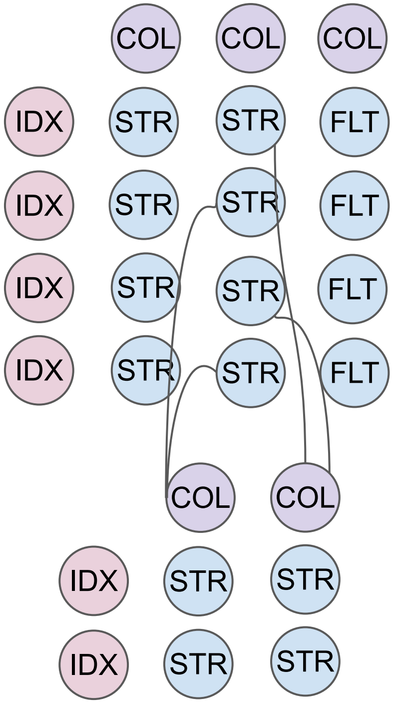
Input

	Date	Category	Expense
0	2018-02-18	Social	98.34
1	2018-02-18	Lunch	245.63
2	2018-02-20	Social	121.89
3	2018-02-20	Lunch	248

Output

	Lunch	Social
2018-02-18	245.63	98.34
2018-02-20	248	121.89

Use *EQUALITY* edges to capture data movement




```
fn_seq = Sequence([pivot, melt, drop...])
```

Key Idea #3 : Graph-Based Encoding

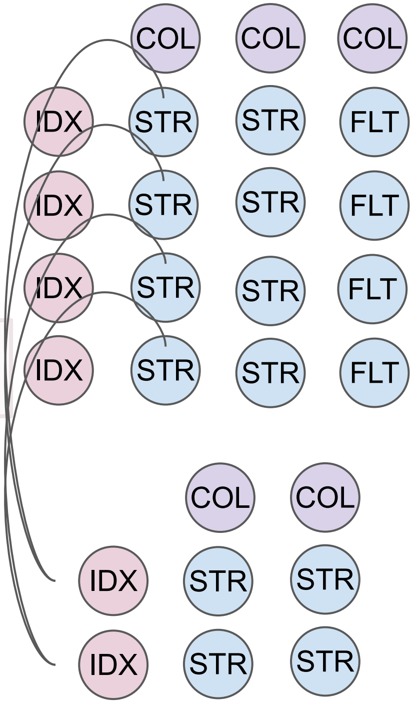
Input

	Date	Category	Expense
0	2018-02-18	Social	98.34
1	2018-02-18	Lunch	245.63
2	2018-02-20	Social	121.89
3	2018-02-20	Lunch	248

Output

	Lunch	Social
2018-02-18	245.63	98.34
2018-02-20	248	121.89

Use *EQUALITY* edges to capture data movement




```
fn_seq = Sequence([pivot, melt, drop...])
```

Key Idea #3 : Graph-Based Encoding

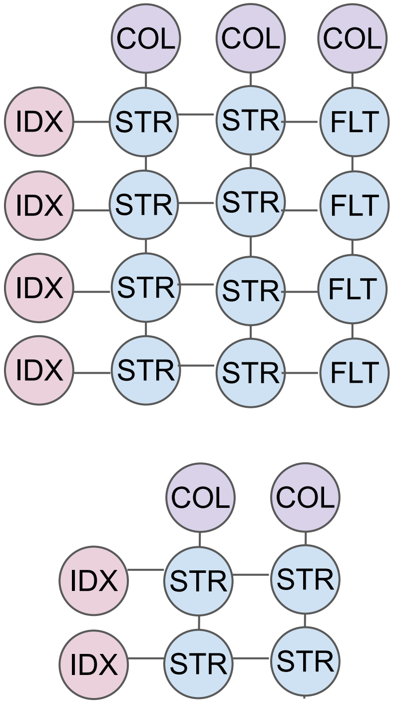
Input

	Date	Category	Expense
0	2018-02-18	Social	98.34
1	2018-02-18	Lunch	245.63
2	2018-02-20	Social	121.89
3	2018-02-20	Lunch	248

Output

	Lunch	Social
2018-02-18	245.63	98.34
2018-02-20	248	121.89

Use *Structural* edges to retain shape information




```
fn_seq = Sequence([pivot, melt, drop...])
```

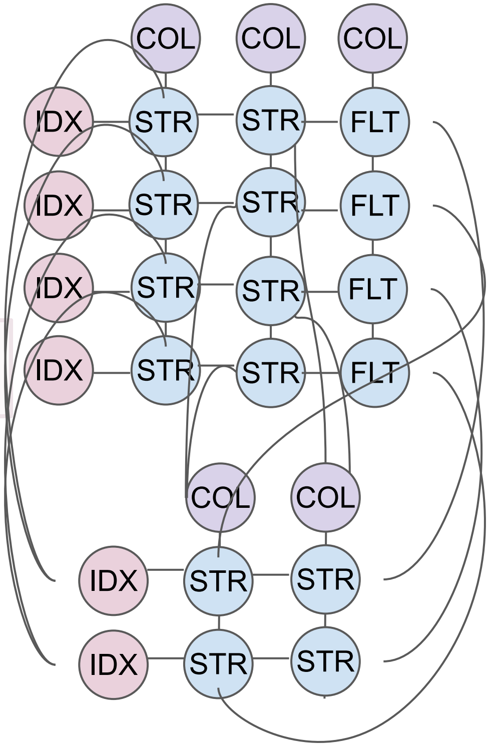
Key Idea #3 : Graph-Based Encoding

Input

	Date	Category	Expense
0	2018-02-18	Social	98.34
1	2018-02-18	Lunch	245.63
2	2018-02-20	Social	121.89
3	2018-02-20	Lunch	248

Output

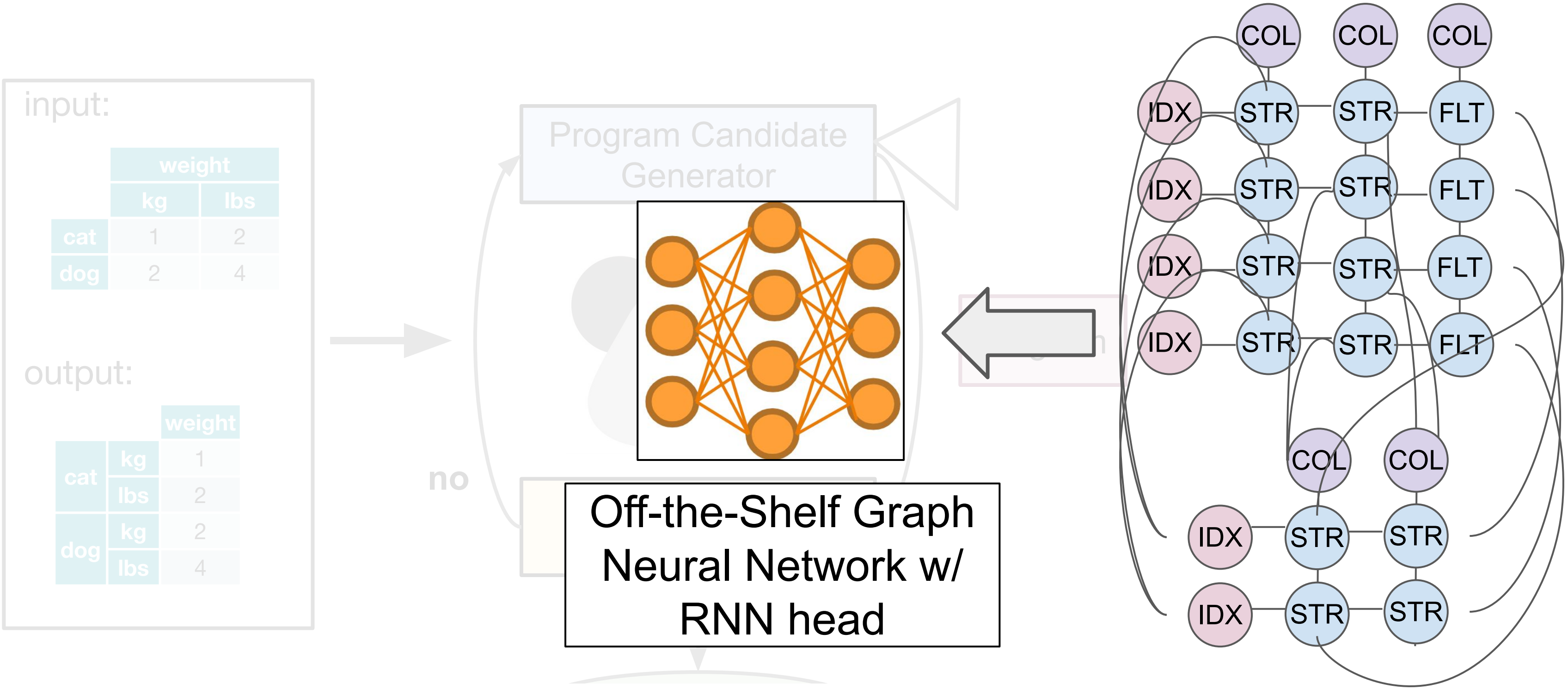
	Lunch	Social
2018-02-18	245.63	98.34
2018-02-20	248	121.89




```
fn_seq = Sequence([pivot, melt, drop...])
```

4. (b) The Model

Use Graph-Neural-Networks for Making Predictions



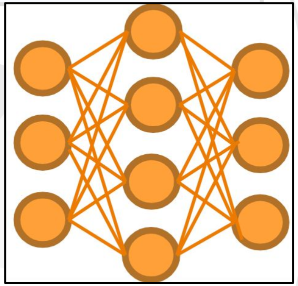

```
fn_seq = Sequence([pivot, melt, drop...])
```

Use Graph-Neural-Networks for Making Predictions

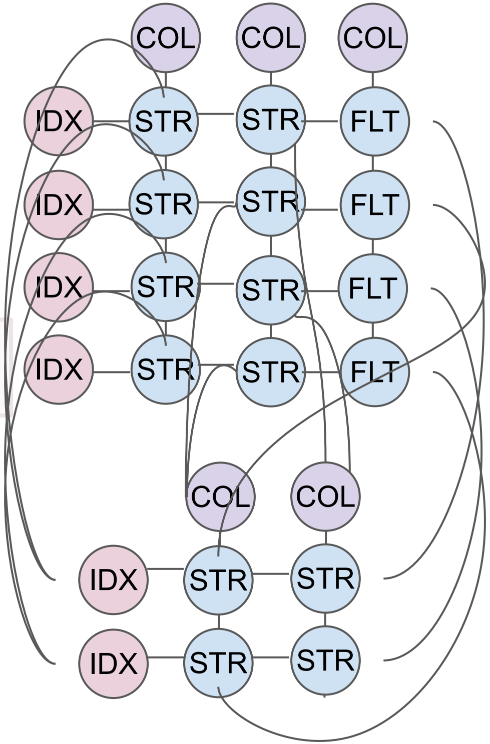
Program Candidate Generator

```
[pivot]: 0.85
[pivot, drop]: 0.10
[groupby, mean]: 0.03
[groupby, max]: 0.01
[stack]: 0.001
```

•
•
•



Off-the-Shelf Graph Neural Network w/ RNN head




```
fn_seq = Sequence([pivot, melt, drop...])
```

4. (c) Training Data

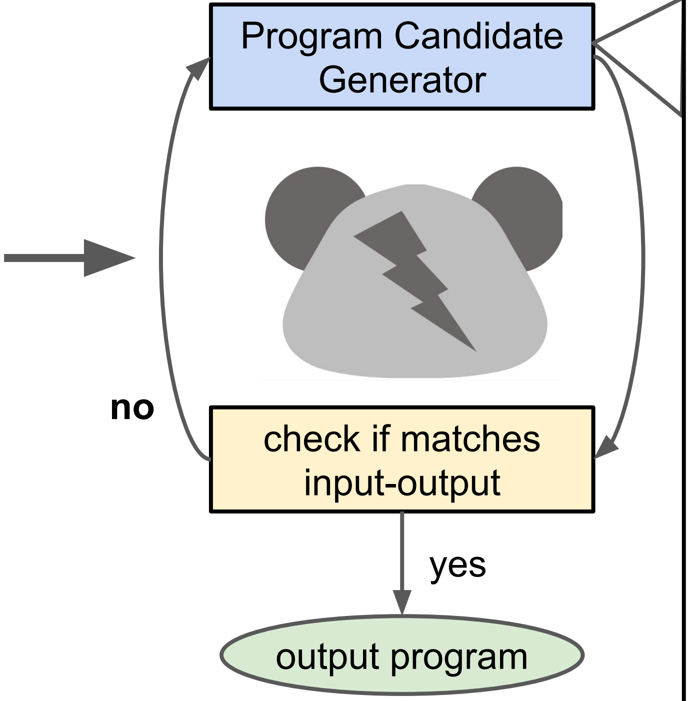
But How to Generate Training Data?

input:

	weight	
	kg	lbs
cat	1	2
dog	2	4

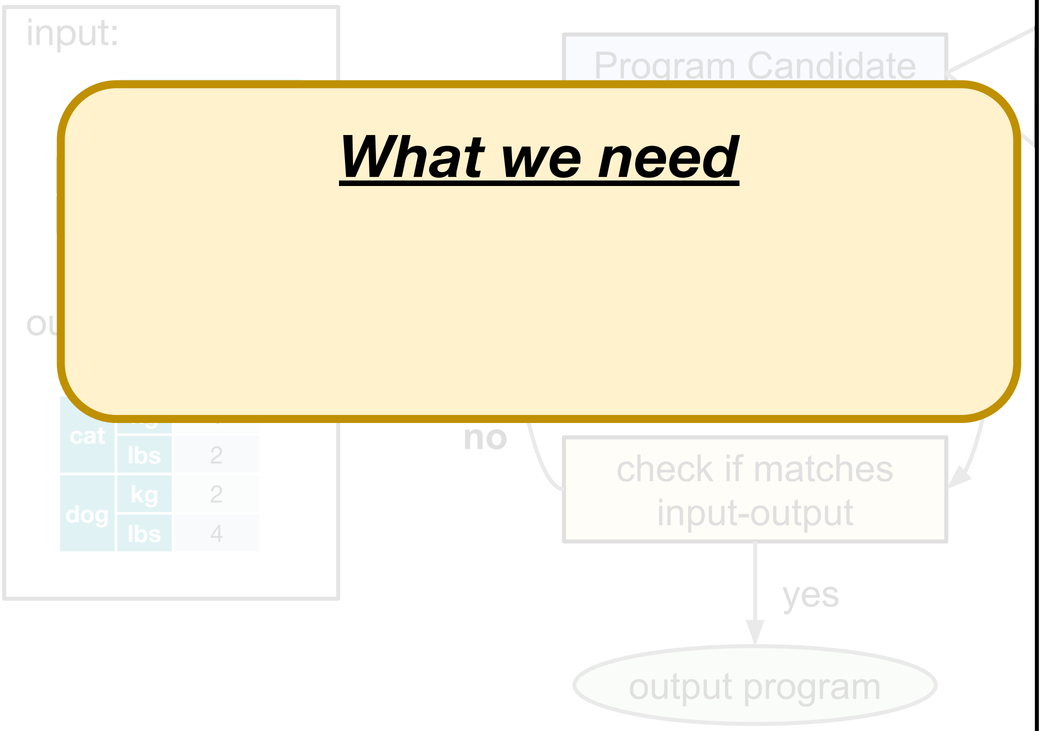
output:

		weight
cat	kg	1
cat	lbs	2
dog	kg	2
dog	lbs	4




```
def generate_program(inputs, output):
  fn_seq = Sequence([pivot, melt, drop...])

  for fn in fn_seq:
    if fn == pivot:
      df = Select(inputs)
      arg_col = Select(df.columns)
      arg_idx = Select(df.columns -
                       {arg_col})
      fn.add_args(df, arg_col, arg_idx)

    elif fn == drop:
      df = Select(inputs)
      arg_ax = Select({0,1})
      arg_lbl = Subset(df.index) if arg_ax
                else Subset(df.columns)
      fn.add_args(df, arg_ax, arg_lbl)

    elif ... :
      # <omitted code...>
    inputs.append(fn.run())
  return fn_seq
```

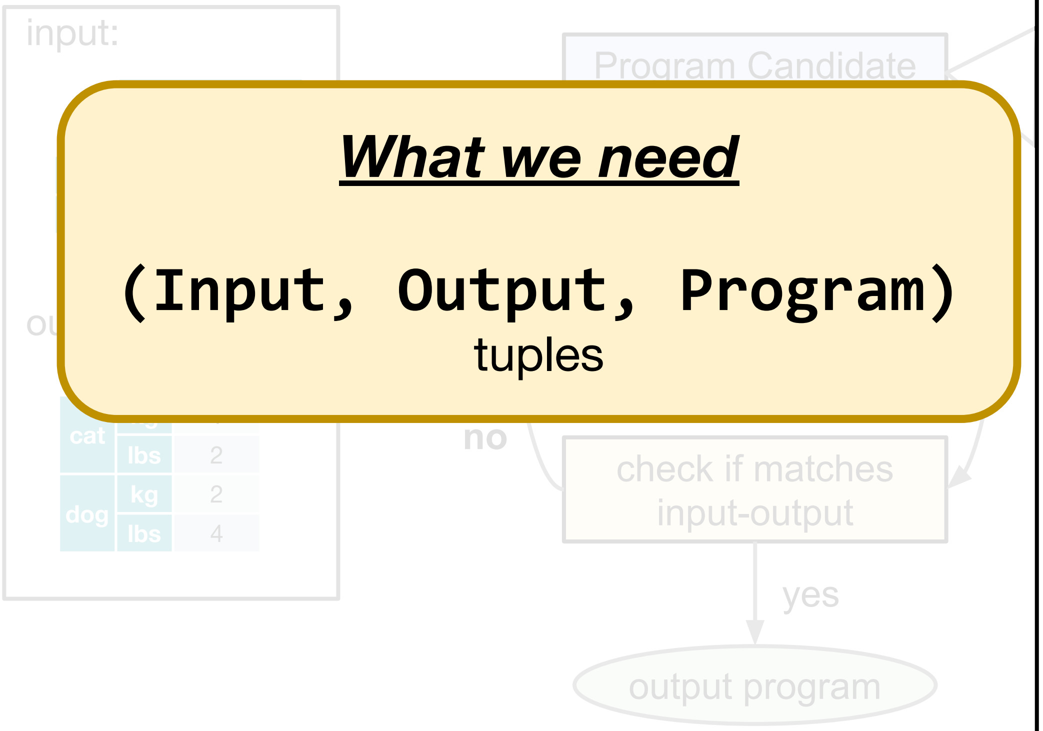
But How to Generate Training Data?



<u>What we need</u>

```
def generate_program(inputs, output):
  fn_seq = Sequence([pivot, melt, drop...])

  for fn in fn_seq:
    if fn == pivot:
      df = Select(inputs)
      arg_col = Select(df.columns)
      arg_idx = Select(df.columns -
                       {arg_col})
      fn.add_args(df, arg_col, arg_idx)

    elif fn == drop:
       df = Select(inputs)
       arg_ax = Select({0,1})
       arg_lbl = Subset(df.index) if arg_ax
                 else Subset(df.columns)
       fn.add_args(df, arg_ax, arg_lbl)

    elif ... :
       # <omitted code...>
    inputs.append(fn.run())
  return fn_seq
```

But How to Generate Training Data?



<u>What we need</u>

(Input, Output, Program)
tuples

```
def generate_program(inputs, output):
  fn_seq = Sequence([pivot, melt, drop...])

  for fn in fn_seq:
    if fn == pivot:
      df = Select(inputs)
      arg_col = Select(df.columns)
      arg_idx = Select(df.columns -
                       {arg_col})
      fn.add_args(df, arg_col, arg_idx)

    elif fn == drop:
       df = Select(inputs)
       arg_ax = Select({0,1})
       arg_lbl = Subset(df.index) if arg_ax
                 else Subset(df.columns)
       fn.add_args(df, arg_ax, arg_lbl)

    elif ... :
       # <omitted code...>
    inputs.append(fn.run())
  return fn_seq
```

But How to Generate Training Data?

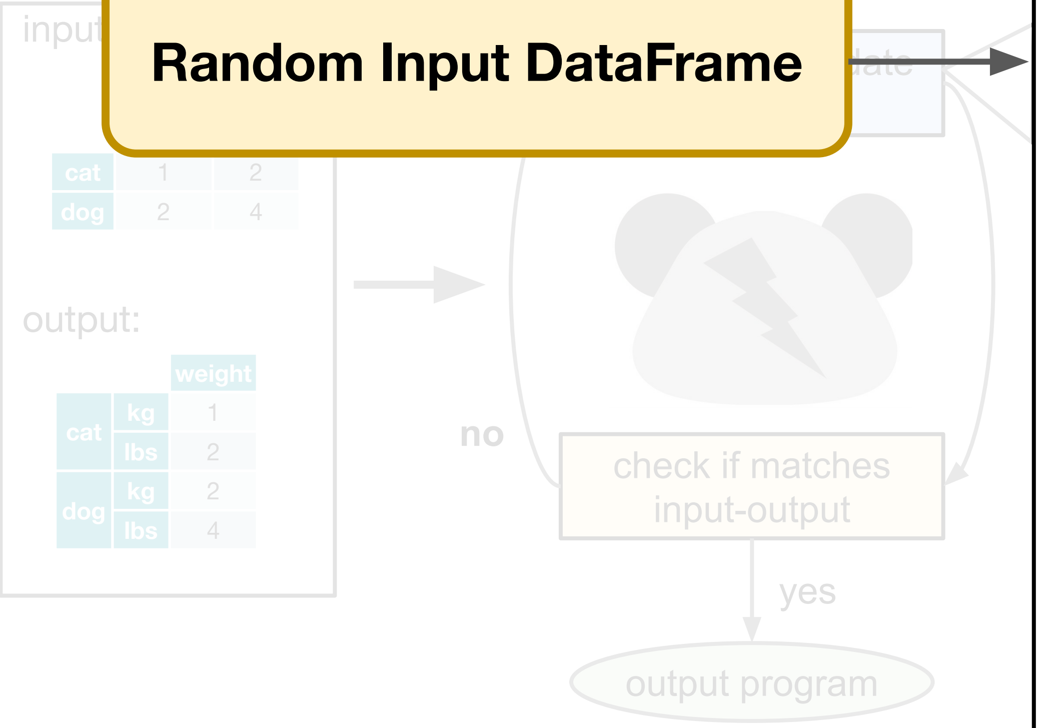
What we need

(Input, Output, Program)
tuples

Reuse the generator!

```
def generate_program(inputs, output):
  fn_seq = Sequence([pivot, melt, drop...])

  for fn in fn_seq:
    if fn == pivot:
      df = Select(inputs)
      arg_col = Select(df.columns)
      arg_idx = Select(df.columns -
                       {arg_col})
      fn.add_args(df, arg_col, arg_idx)

    elif fn == drop:
       df = Select(inputs)
       arg_ax = Select({0,1})
       arg_lbl = Subset(df.index) if arg_ax
                 else Subset(df.columns)
       fn.add_args(df, arg_ax, arg_lbl)

    elif ... :
       # <omitted code...>
    inputs.append(fn.run())
  return fn_seq
```

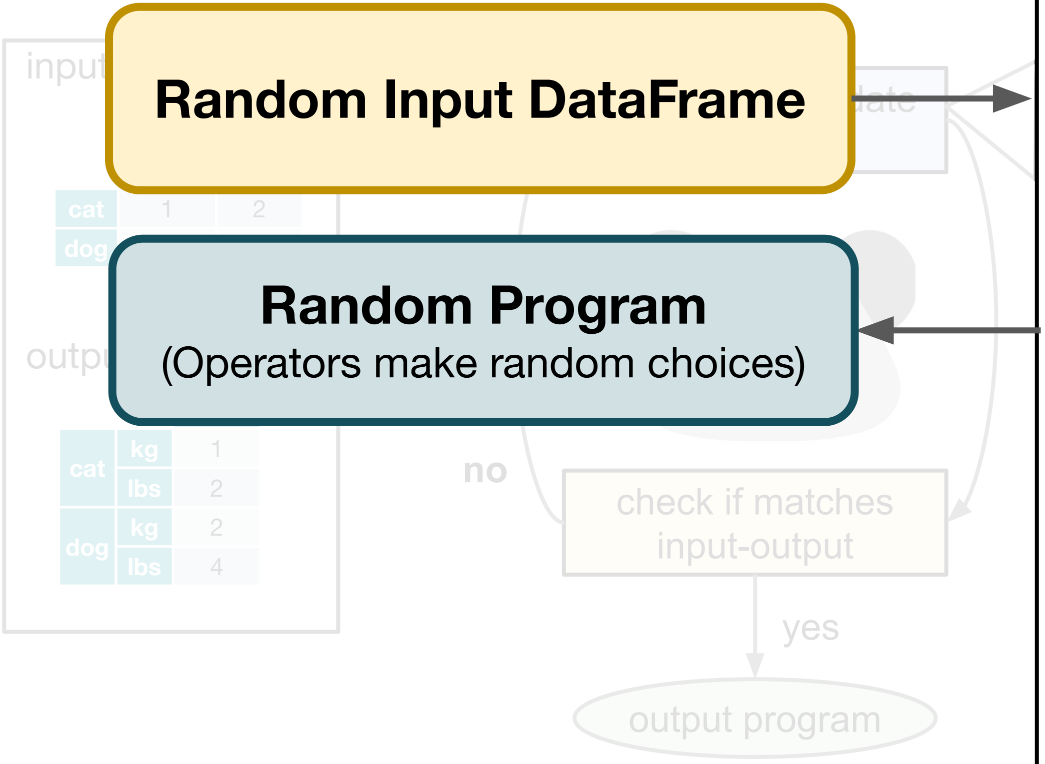
Key Idea #4 : Generate Random Synthetic Data



Random Input DataFrame

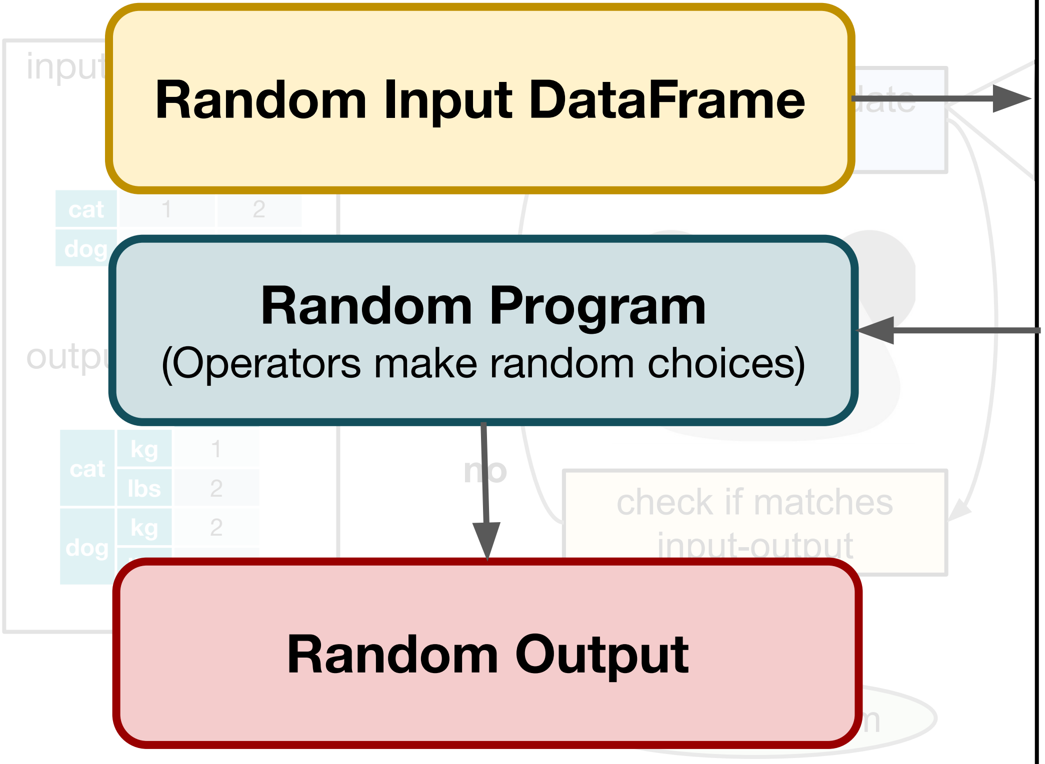
```
def generate_program(inputs, output):
  fn_seq = Sequence([pivot, melt, drop...])

  for fn in fn_seq:
    if fn == pivot:
      df = Select(inputs)
      arg_col = Select(df.columns)
      arg_idx = Select(df.columns -
                       {arg_col})
      fn.add_args(df, arg_col, arg_idx)

    elif fn == drop:
      df = Select(inputs)
      arg_ax = Select({0,1})
      arg_lbl = Subset(df.index) if arg_ax
                else Subset(df.columns)
      fn.add_args(df, arg_ax, arg_lbl)

    elif ... :
      # <omitted code...>
    inputs.append(fn.run())
  return fn_seq
```

Key Idea #4 : Generate Random Synthetic Data




```
def generate_program(inputs, output):
  fn_seq = Sequence([pivot, melt, drop...])

  for fn in fn_seq:
    if fn == pivot:
      df = Select(inputs)
      arg_col = Select(df.columns)
      arg_idx = Select(df.columns -
                       {arg_col})
      fn.add_args(df, arg_col, arg_idx)

    elif fn == drop:
      df = Select(inputs)
      arg_ax = Select({0,1})
      arg_lbl = Subset(df.index) if arg_ax
                else Subset(df.columns)
      fn.add_args(df, arg_ax, arg_lbl)

    elif ... :
      # <omitted code...>
    inputs.append(fn.run())
  return fn_seq
```

Key Idea #4 : Generate Random Synthetic Data




```
def generate_program(inputs, output):
  fn_seq = Sequence([pivot, melt, drop...])

  for fn in fn_seq:
    if fn == pivot:
      df = Select(inputs)
      arg_col = Select(df.columns)
      arg_idx = Select(df.columns -
                       {arg_col})
      fn.add_args(df, arg_col, arg_idx)

    elif fn == drop:
      df = Select(inputs)
      arg_ax = Select({0,1})
      arg_lbl = Subset(df.index) if arg_ax
                else Subset(df.columns)
      fn.add_args(df, arg_ax, arg_lbl)

    elif ... :
      # <omitted code...>
    inputs.append(fn.run())
  return fn_seq
```

Evaluation

Encouraging Results on Real Stackoverflow Benchmarks

	Depth	Candidates Explored	Sequences Explored	Solved	Time(s)
SO_11881165	1	15	1	Y	0.54
SO_11941492	1	783	8	Y	12.55
SO_13647222	1	5	1	Y	3.32
SO_18172851	1	-	-	N	-
SO_49583055	1	-	-	N	-
SO_49592930	1	2	1	Y	1.1
SO_49572546	1	3	1	Y	1.1
SO_13261175	1	39537	18	Y	300.2
SO_13793321	1	92	1	Y	4.16
SO_14085517	1	10	1	Y	2.24
SO_11418192	2	158	1	Y	0.71
SO_49567723	2	1684022	2	Y	753.1
SO_13261691	2	65	1	Y	2.96
SO_13659881	2	2	1	Y	1.38
SO_13807758	2	711	2	Y	7.21
SO_34365578	2	-	-	N	-
SO_10982266	3	-	-	N	-
SO_11811392	3	-	-	N	-
SO_49581206	3	-	-	N	-
SO_12065885	3	924	1	Y	0.9
SO_13576164	3	22966	5	Y	339.25
SO_14023037	3	-	-	N	-
SO_53762029	3	27	1	Y	1.9
SO_21982987	3	8385	10	Y	30.8
SO_39656670	3	-	-	N	-
SO_23321300	3	-	-	N	-

Collected 26 Real-World Stack-Overflow Benchmarks

Could solve **17/26 (65%)** Benchmarks

Encouraging Results on Real Stackoverflow Benchmarks

	Depth	Candidates Explored	Sequences Explored	Solved	Time(s)
SO_11881165	1	15	1	Y	0.54
SO_11941492	1	783	8	Y	12.55
SO_13647222	1	5	1	Y	3.32
SO_18172851	1	-	-	N	-
SO_49583055	1	-	-	N	-
SO_49592					1.1
SO_49572					1
SO_13261					0.2
SO_13793					16
SO_14085					24
SO_11418					71
SO_49567					3.1
SO_13261					96
SO_13659					38
SO_13807					21
SO_343655					
SO_10982266	3	-	-	N	-
SO_11811392	3	-	-	N	-
SO_49581206	3	-	-	N	-
SO_12065885	3	924	1	Y	0.9
SO_13576164	3	22966	5	Y	339.25
SO_14023037	3	-	-	N	-
SO_53762029	3	27	1	Y	1.9
SO_21982987	3	8385	10	Y	30.8
SO_39656670	3	-	-	N	-
SO_23321300	3	-	-	N	-

90% of Accepted Answers on StackOverflow contain **upto 3 functions**

Collected 26 Real-World Stack-Overflow Benchmarks

Could solve **17/26 (65%)** Benchmarks

Can find programs containing **three**-function sequences

Encouraging Results on Real Stackoverflow Benchmarks

	Depth	Candidates Explored	Sequences Explored	Solved	Time(s)
SO_11881165	1	15	1	Y	0.54
SO_11941492	1	783	8	Y	12.55
SO_13647222	1	5	1	Y	3.32
SO_18172851	1	-	-	N	-
SO_49583055	1	-	-	N	-
SO_49592930	1	2	1	Y	1.1
SO_49572546	1	3	1	Y	1.1
SO_13261175	1	39537	18	Y	300.2
SO_13793321	1	92	1	Y	4.16
SO_14085517	1	10	1	Y	2.24
SO_11418192	2	158	1	Y	0.71
SO_49567723	2	1684022	2	Y	753.1
SO_13261691	2	65	1	Y	2.96
SO_13659881	2	2	1	Y	1.38
SO_13807758	2	711	2	Y	7.21
SO_34365578	2	-	-	N	-
SO_10982266	3	-	-	N	-
SO_11811392	3	-	-	N	-
SO_49581206	3	-	-	N	-
SO_12065885	3	924	1	Y	0.9
SO_13576164	3	22966	5	Y	339.25
SO_14023037	3	-	-	N	-
SO_53762029	3	27	1	Y	1.9
SO_21982987	3	8385	10	Y	30.8
SO_39656670	3	-	-	N	-
SO_23321300	3	-	-	N	-

Collected 26 Real-World Stack-Overflow Benchmarks

Could solve **17/26 (65%)** Benchmarks

Most solutions found in top-10 function sequences explored

Discussion

Discussion

Overcomes limitations of Grammars/DSLs

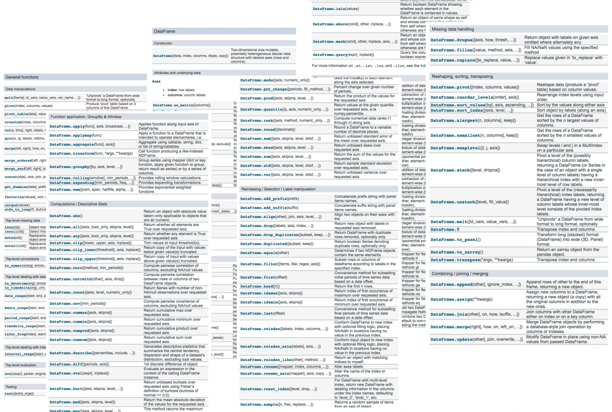
https://rbavishi.github.io/autopandas

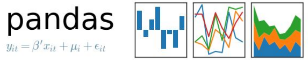
Combining many small models

<u>Main Contribution / Idea</u>

Generators as a Unified Abstraction for Search Space + Algorithm

AutoPandas Summary





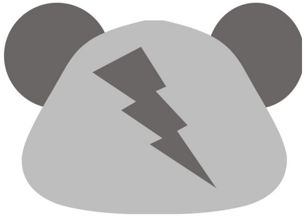

```
def generate_program(inputs, output):
  fn_seq = Sequence([pivot, melt, drop...])

  for fn in fn_seq:
    if fn == pivot:
      df = Select(inputs)
      arg_col = Select(df.columns)
      arg_idx = Select(df.columns -
                                  {arg_col})
      fn.add_args(df, arg_col, arg_idx)

    elif fn == drop:
      df = Select(inputs)
      arg_ax = Select({0,1})
      arg_lbl = Subset(df.index) if arg_ax
              else Subset(df.columns)
      fn.add_args(df, arg_ax, arg_lbl)

    elif ... :
      # <omitted code...>
    inputs.append(fn.run())
  return fn_seq
```

Encouraging Results on Real Stackoverflow Benchmarks

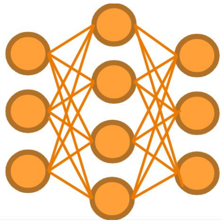
	Depth	Candidates Explored	Sequences Explored	Solved	Time(s)
SO_11881165	1	15	1	Y	0.54
SO_11941492	1	783	8	Y	12.55
SO_13647222	1	5	1	Y	3.32
SO_18172851	1	-	-	N	-
SO_49583055	1	-	-	N	-
SO_49592930	1	2	1	Y	1.1
SO_49572546	1	3	1	Y	1.1
SO_13261175	1	39537	18	Y	300.2
SO_13793321	1	92	1	Y	4.16
SO_14085517	1	10	1	Y	2.24
SO_11418192	2	158	1	Y	0.71
SO_49567723	2	1684022	2	Y	753.1
SO_13261691	2	65	1	Y	2.96
SO_13659881	2	2	1	Y	1.38
SO_13807758	2	711	2	Y	7.21
SO_34365578	2	-	-	N	-
SO_10982266	3	-	-	N	-
SO_11811392	3	-	-	N	-
SO_49581206	3	-	-	N	-
SO_12065885	3	924	1	Y	0.9
SO_13576164	3	22966	5	Y	339.25
SO_14023037	3	-	-	N	-
SO_53762029	3	27	1	Y	1.9
SO_21982987	3	8385	10	Y	30.8
SO_39656670	3	-	-	N	-
SO_23321300	3	-	-	N	-

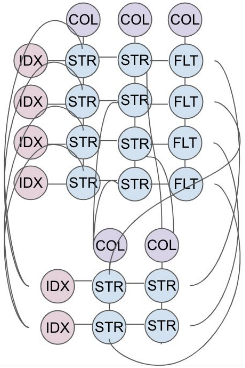
Collected 26 Real-World Stack-Overflow Benchmarks

Could solve **17/26 (65%)** Benchmarks



https://autopandas.io





Demo

https://rbavishi.github.io/autopandas

Video Link: https://www.youtube.com/watch?v=DYwC3XAC9_0

Why Input-Output Examples are Not Ideal

Information Loss + Tediousness

Input-Output Examples are **Problematic** for Table Transformations

Problem #1

Readily available information is lost

Loss in performance and increased chance of overfitting

A synthesis engine must determine how this being computed...

input

	Type	Low	High
0	Pants	50	70
1	Pants	100	190
2	Shirts	80	110

output

	Type	Avg
0	Pants	$\boxed{102.5}$
1	Shirts	95

102.5 is the average of 50, 70, 100, 190

This information is known to the user!

Input-Output Examples are **Problematic** for Table Transformations

Problem #2

*Providing Examples can be **Tedious***

input

	Type	Low	High
0	Pants	50	70
1	Pants	100	190
2	Shirts	80	110

⋮

100 more rows

output ???

Research

AutoPandas: Neural-Backed Generators for Program Synthesis
***Rohan Bavishi**, Caroline Lemieux, Roy Fox, Koushik Sen, Ion Stoica*
OOPSLA 2019

Gauss: Program Synthesis by Reasoning Over Graphs
***Rohan Bavishi**, Caroline Lemieux, Koushik Sen, Ion Stoica*
OOPSLA 2021

VizSmith: Automated Visualization Synthesis by Mining Data-Science Notebooks
***Rohan Bavishi**, Shadaj Laddad, Hiroaki Yoshida, Mukul R. Prasad, Koushik Sen*
ASE 2021

Gauss at a Glance

Capturing Partial Examples and User Intent as a Graph

Gauss Algorithm

Graph-Based Inductive Reasoning to Search Faster

Evaluation

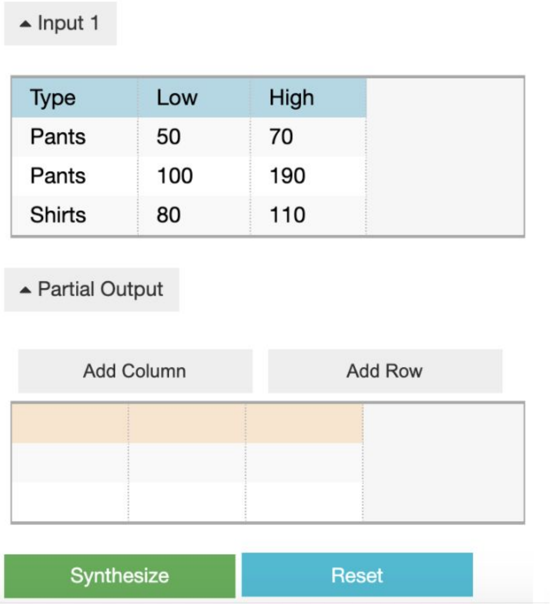
https://www.youtube.com/watch?v=
Z6pZM1RP1OA&t=1s

Demo

https://github.com/rbavishi/gauss-oopsla-2021

Video Link: https://www.youtube.com/watch?v=Z6pZM1RP1OA

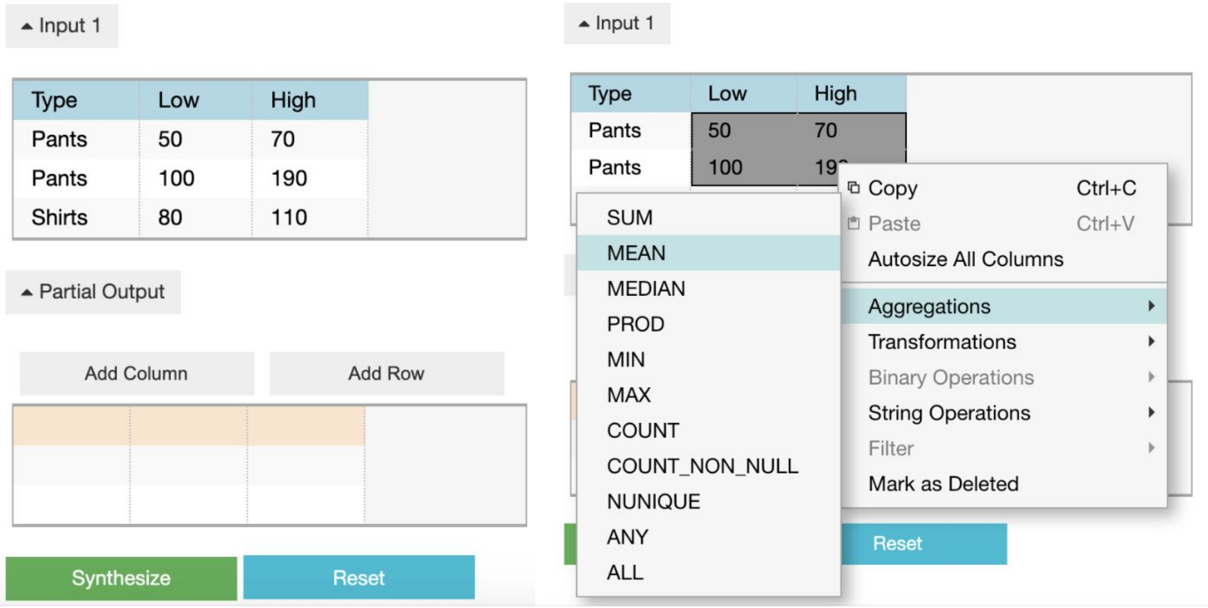
Gauss Uses a **Dedicated UI** to Capture Interaction



Type	Low	High
Pants	50	70
Pants	100	190
Shirts	80	110

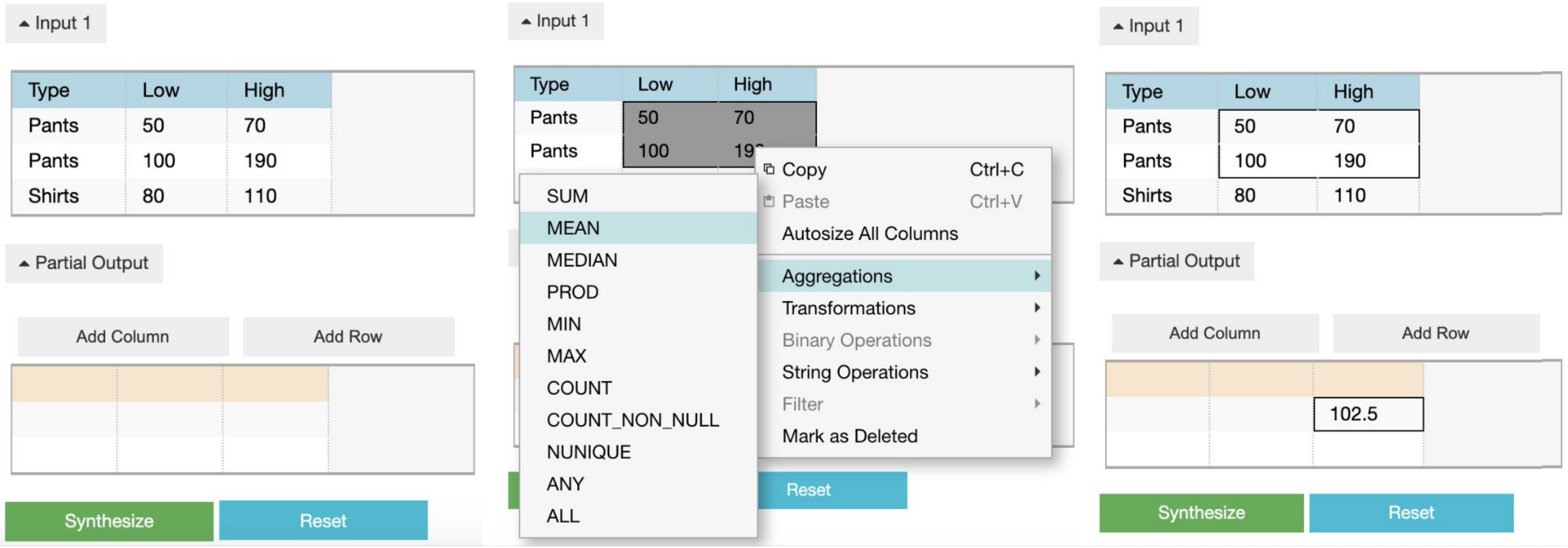
User loads the input dataframe

Gauss Uses a **Dedicated UI** to Capture Interaction



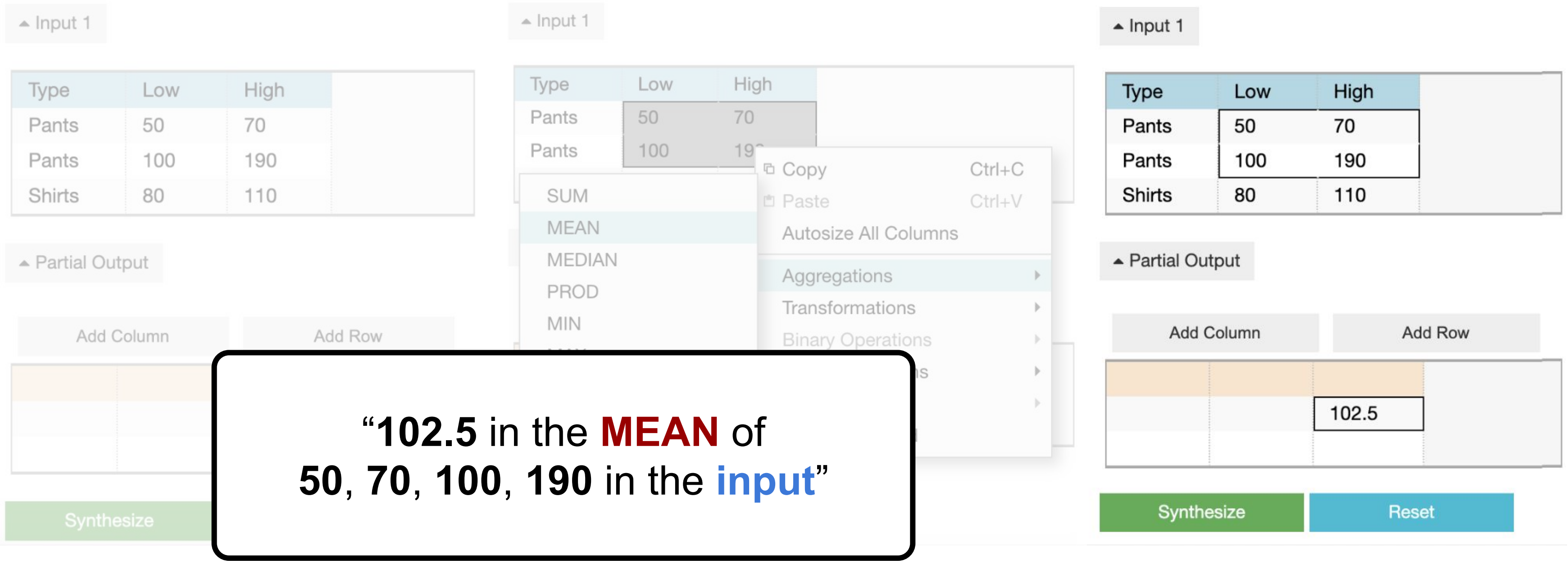
User selects the appropriate cells and the operation to perform.
The result is copied onto the clipboard

Gauss Uses a **Dedicated UI** to Capture Interaction



User pastes the value into the partial output editor.

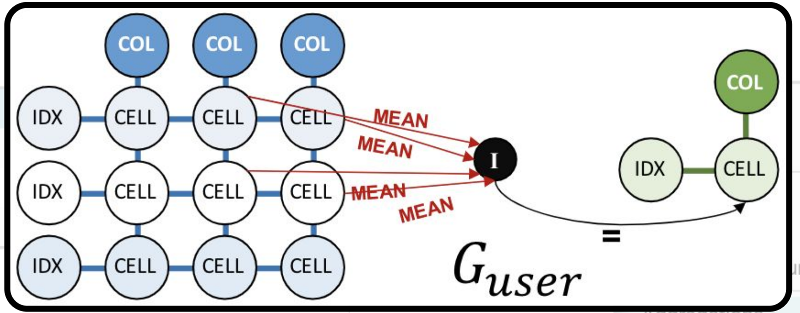
Gauss Represents the Interaction as a **Graph**



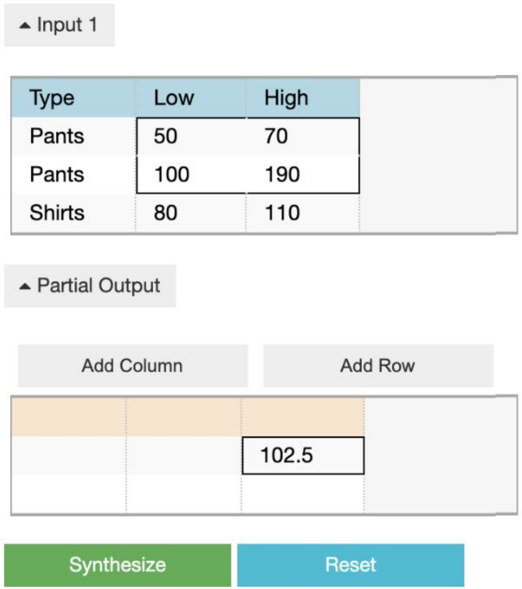
"**102.5** in the **MEAN** of
50, **70**, **100**, **190** in the **input**"

User pastes the value into the partial output editor.

Gauss Represents the Interaction as a **Graph**

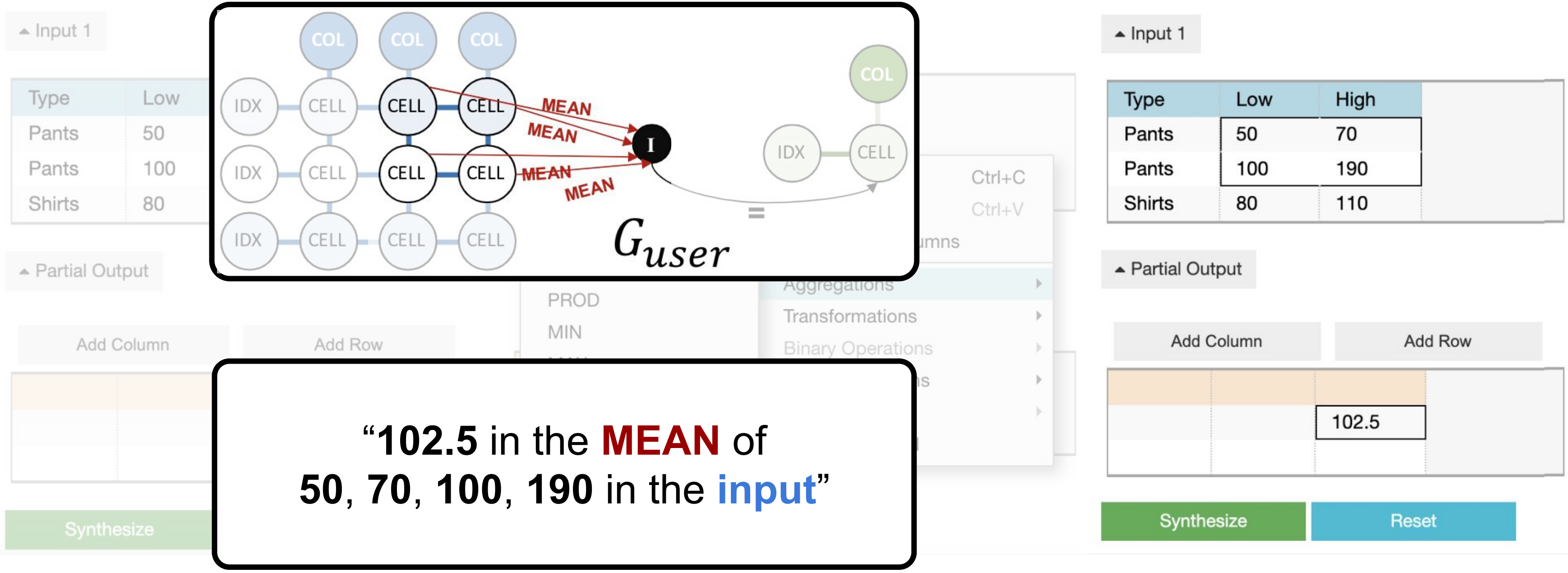


“102.5 in the **MEAN** of
50, **70**, **100**, **190** in the **input”**



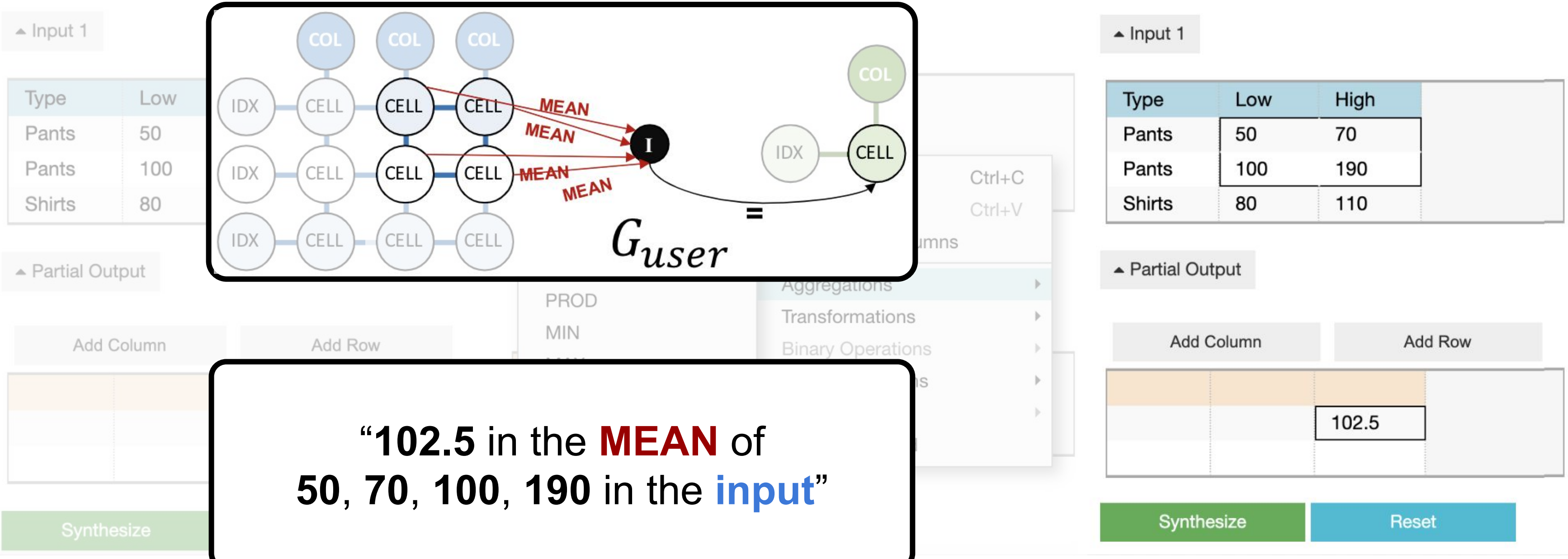
User pastes the value into the partial output editor.

Gauss Represents the Interaction as a **Graph**



User pastes the value into the partial output editor.

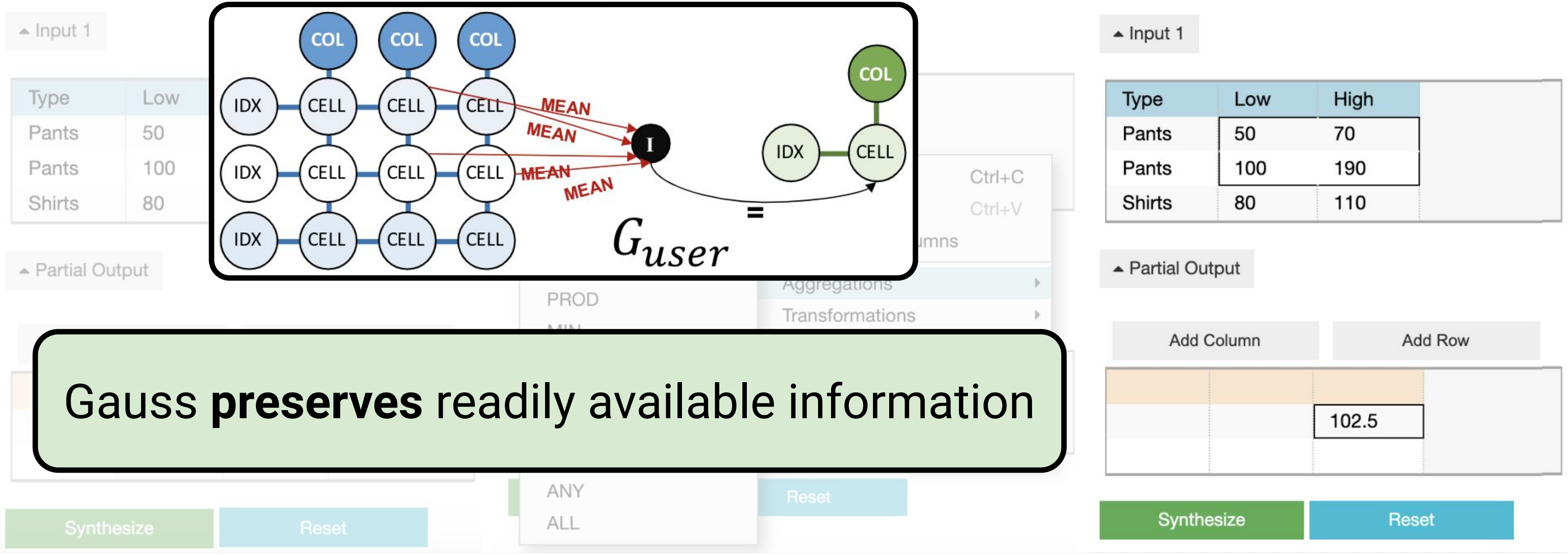
Gauss Represents the Interaction as a **Graph**



“**102.5** in the **MEAN** of
50, **70**, **100**, **190** in the **input**”

User pastes the value into the partial output editor.

Gauss Represents the Interaction as a **Graph**



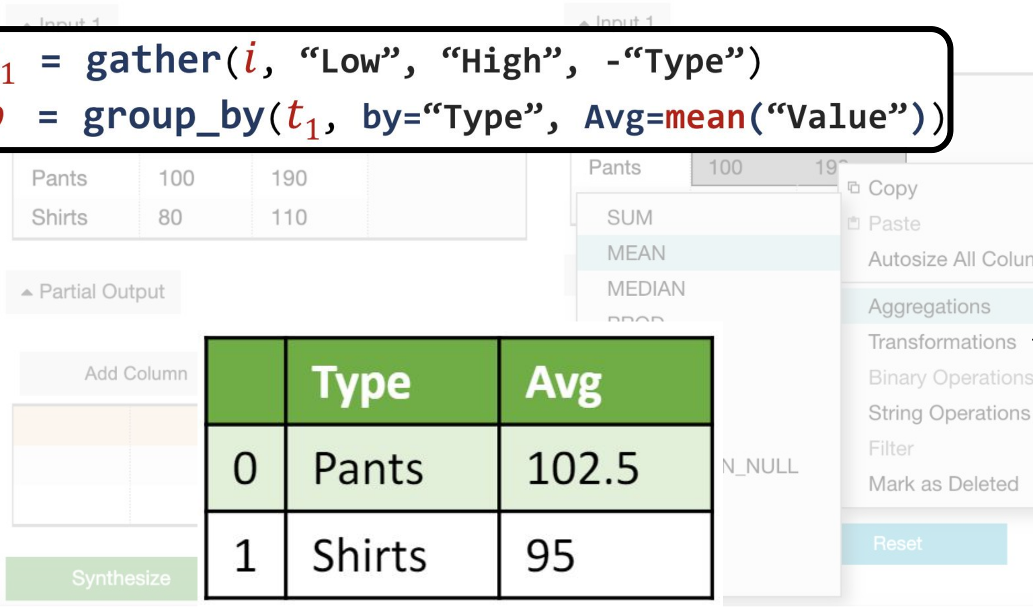
Gauss **preserves** readily available information

User pastes the value into the partial output editor.

Gauss Accepts **Partial Outputs**

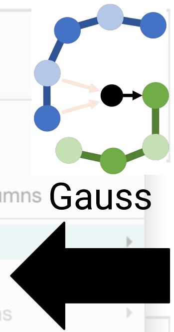
Synthesized R Program

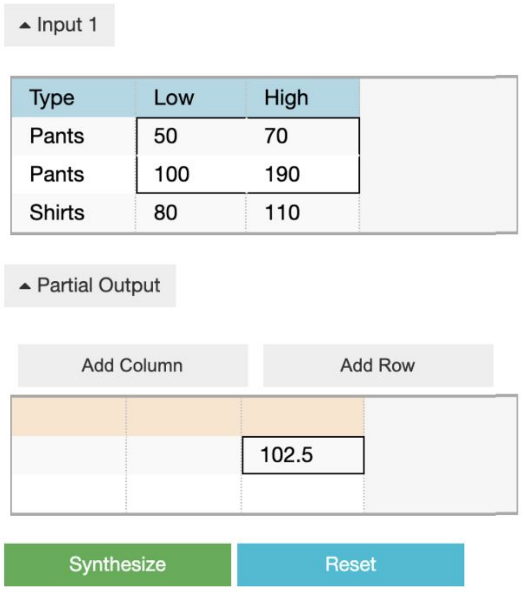
t_1 = gather(i, “Low”, “High”, -“Type”)
o = group_by(t_1, by=“Type”, Avg=mean(“Value”))



	Type	Avg
0	Pants	102.5
1	Shirts	95

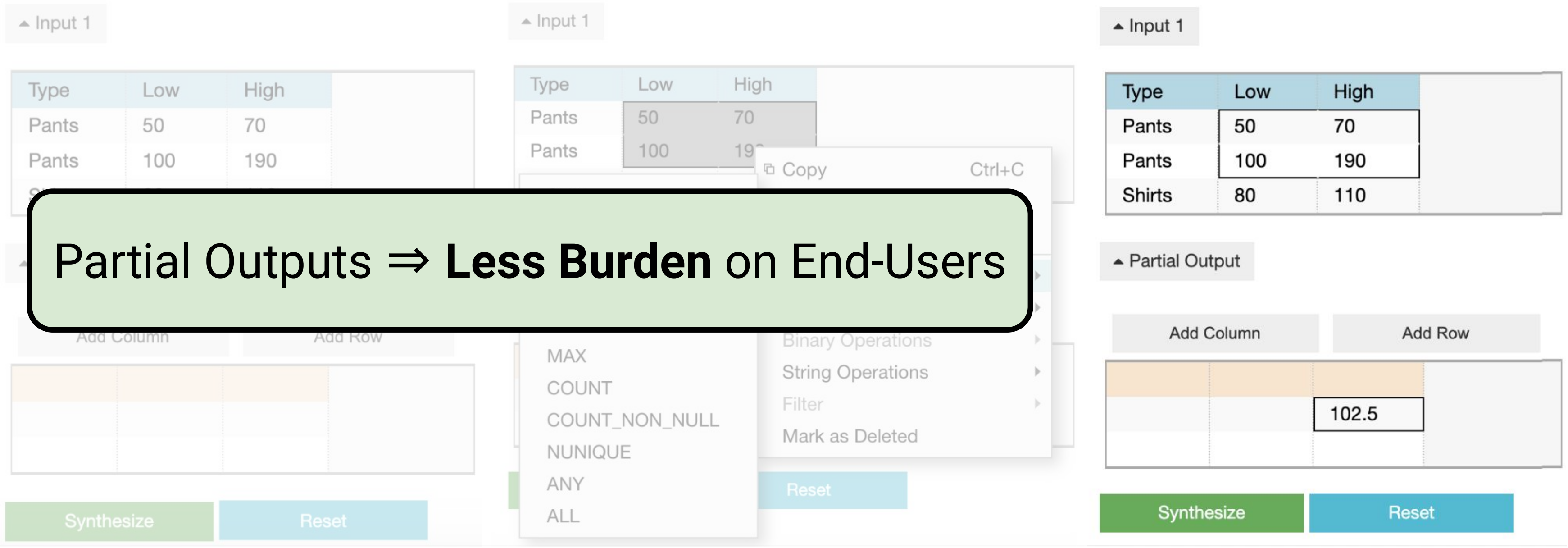
Full Output of Program





User can now click Synthesize

Gauss Accepts **Partial Outputs**



Partial Outputs ⇒ **Less Burden** on End-Users

User can now click Synthesize

The Graph-Based Synthesis Specification

Input Table

	Type	Low	High
0	Pants	50	70
1	Pants	100	190
2	Shirts	80	110

Partial Output

		102.5

User Intent:
“**102.5** in the **MEAN** of
50, **70**, **100**, **190** in the **input**”

The Graph-Based Synthesis Specification

Input Table

	Type	Low	High
0	Pants	50	70
1	Pants	100	190
2	Shirts	80	110

Partial Output

		102.5

Partial Output ***contained*** *in Program Output*

User Intent:
“**102.5** in the **MEAN** of
50, **70**, **100**, **190** in the **input**”

```
t1 = gather(i, “Low”, “High”, -“Type”)
o  = group_by(t1, by=“Type”, Avg=mean(“Value”))
```

	Type	Avg
0	Pants	102.5
1	Shirts	95

Solution Program **P**
returned by Gauss

Its Output Table

The Graph-Based Synthesis Specification

Input Table

	Type	Low	High
0	Pants	50	70
1	Pants	100	190
2	Shirts	80	110

Partial Output

		102.5

Partial Output ***contained*** *in Program Output*

```
t1 = gather(i, "Low", "High", -"Type")
o  = group_by(t1, by="Type", Avg=mean("Value"))
```

Solution Program **P**
returned by Gauss

	Type	Avg
0	Pants	102.5
1	Shirts	95

Its Output Table

User Intent:
"**102.5** in the **MEAN** of
50, **70**, **100**, **190** in the **input**"

must be consistent with:

Behavior of the Program **P**

The Graph-Based Synthesis Specification

Input Table

	Type	Low	High
0	Pants	50	70
1	Pants	100	190
2	Shirts	80	110

Partial Output

		102.5

Partial Output ***contained*** *in Program Output*

```
t1 = gather(i, "Low", "High", -"Type")
o  = group_by(t1, by="Type", Avg=mean("Value"))
```

	Type	Avg
0	Pants	102.5
1	Shirts	95

Solution Program **P**
returned by Gauss

Its Output Table

User Intent Graph

must be a subgraph of:

Graph Abstraction of Program P

The Graph-Based Synthesis Specification

The graph abstraction of P captures its behavior on the given input as a graph

User Intent Graph

must be a subgraph of:

Partial Output ***contained*** *in Program Output*

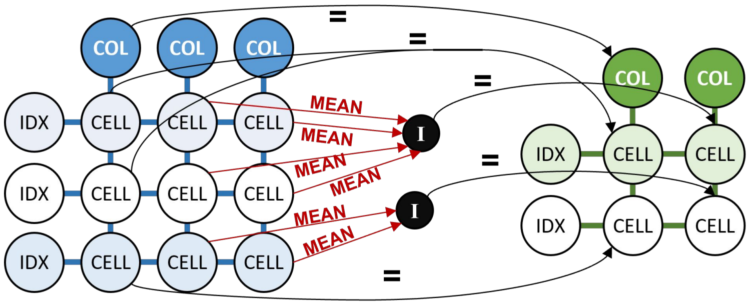
	Type	Low	High
0	Pants	50	70
1	Pants	100	190
2	Shirts	80	110

Input to P

	Type	Avg
0	Pants	102.5
1	Shirts	95

Output of P

t_1 = gathe
o = group_
'value'



Graph Abstraction **of P**

Solution Program ***P***
returned by Gauss

Its Output Table

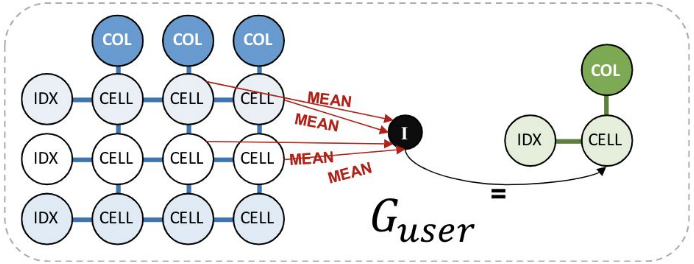
The Graph-Based Synthesis Specification

Input Table

	Type	Low	High
0	Pants	50	70
1	Pants	100	190

Partial Output

		102.5

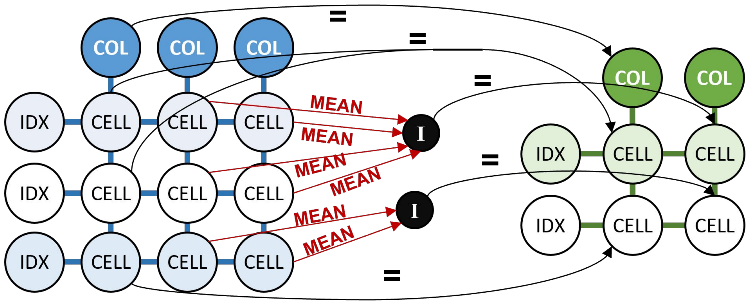


G_{user} *must be a* ***subgraph*** *of the graph abstraction of* **P**

$\subseteq$

```
t1 = gather(i, "Low", "High", -"Type")
o  = group_by(t1, by="Type", Avg=mean("Value"))
```

	Type	Avg
0	Pants	102.5
1	Shirts	95



Solution Program **P**
returned by Gauss

Its Output Table

Graph Abstraction of **P**

The Graph-Based Synthesis Specification

Input Table

	Type	Low	High
0	Pants	50	70
1	Pants	100	190

Partial Output

		102.5

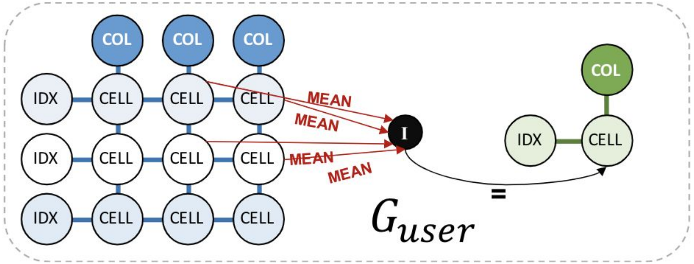
G_{user} *must be a* ***subgraph*** *of the graph abstraction of* **P**

```
t1 = gather(i, "Low", "High", -"Type")
o  = group_by(t1, by="Type", Avg=mean("Value"))
```

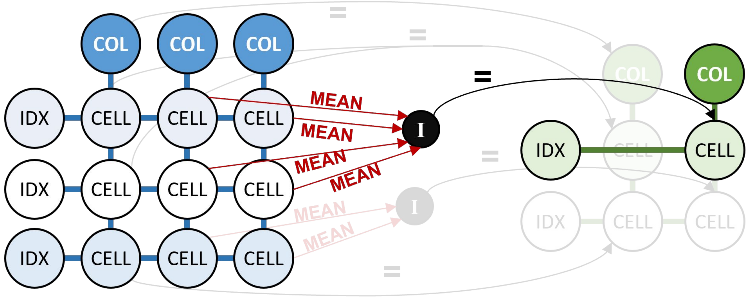
	Type	Avg
0	Pants	102.5
1	Shirts	95

Solution Program **P**
returned by Gauss

Its Output Table



$\subseteq$



Graph Abstraction of **P**

The Graph-Based Synthesis Specification

Input Table

	Type	Low	High
0	Pants	50	70
1	Pants	100	190

Partial Output

		102.5

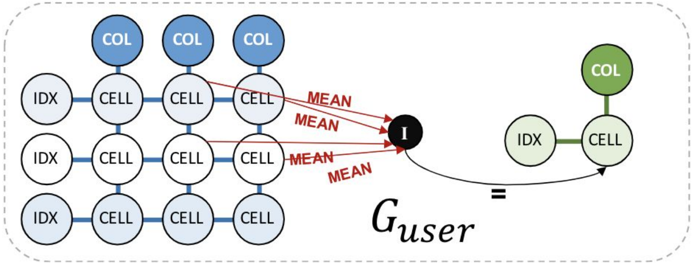
Enforces a stronger match with the user's intent

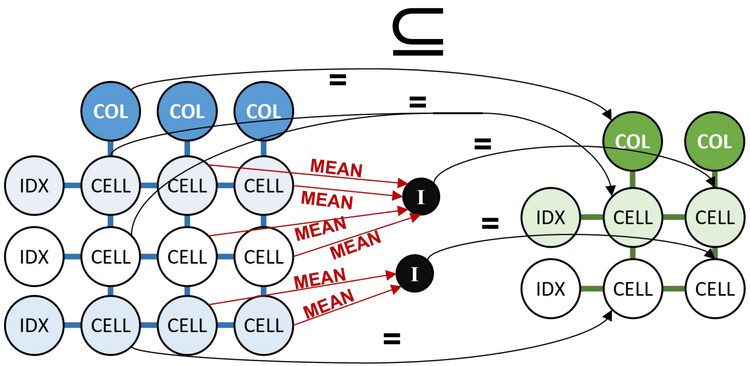
```
t1 = gather(i, "Low", "High", -"Type")
o  = group_by(t1, by="Type", Avg=mean("Value"))
```

Solution Program **P**
returned by Gauss

	Type	Avg
0	Pants	102.5
1	Shirts	95

Its Output Table





Graph Abstraction of **P**

The Graph-Based Synthesis Specification

Input Table

	Type	Low	High
0	Pants	50	70
1	Pants	100	190

Partial Output

		102.5

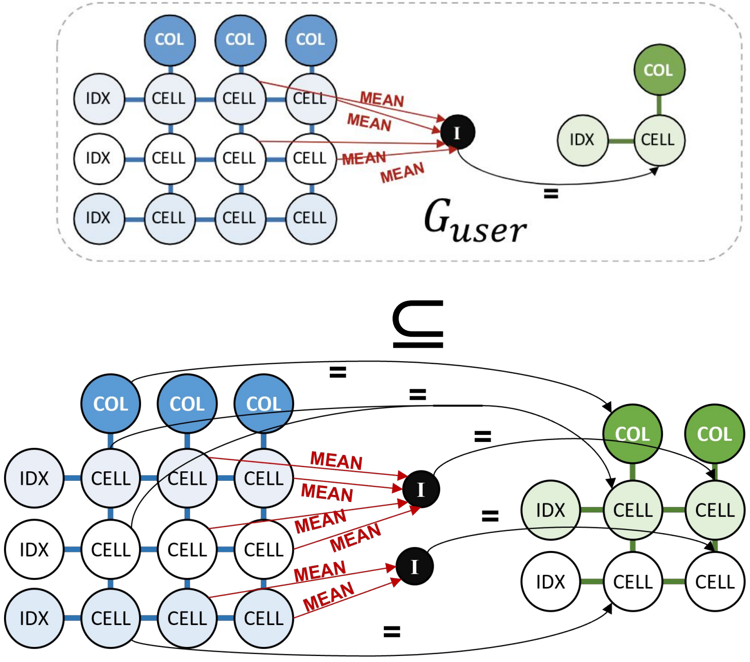
How to obtain the graph abstraction of a program?

```
t1 = gather(i, “Low”, “High”, -“Type”)
o  = group_by(t1, by=“Type”, Avg=mean(“Value”))
```

Solution Program **P**
returned by Gauss

	Type	Avg
0	Pants	102.5
1	Shirts	95

Its Output Table



Graph Abstraction **of P**

The Graph-Based Synthesis Specification

Augment generators from before!
Modify generators for individual functions to return graphs along with func. call args

1	Pants	100	190
2	Shirts	80	110

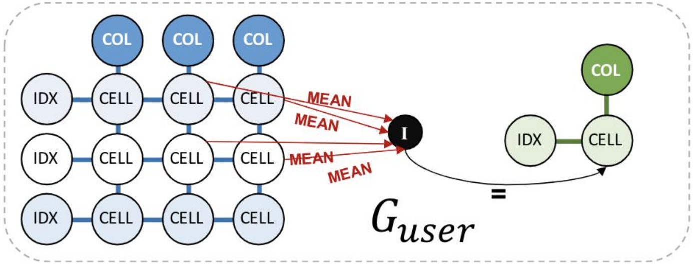
Graph abstractions for each function call in the program P

	Type	Avg

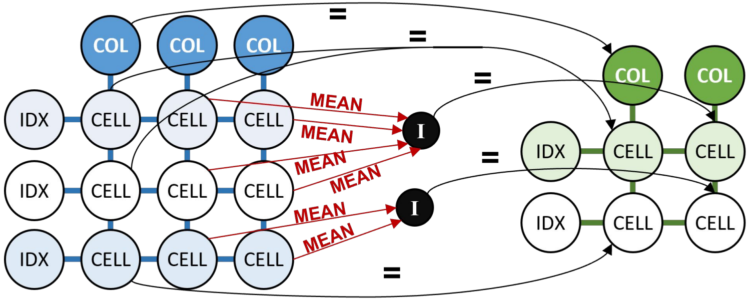
t_1 = gather(i, “Low”, “High”, -“Type”)
o = gr

Composed together to produce the graph abstraction of the full program P

returned by Gauss



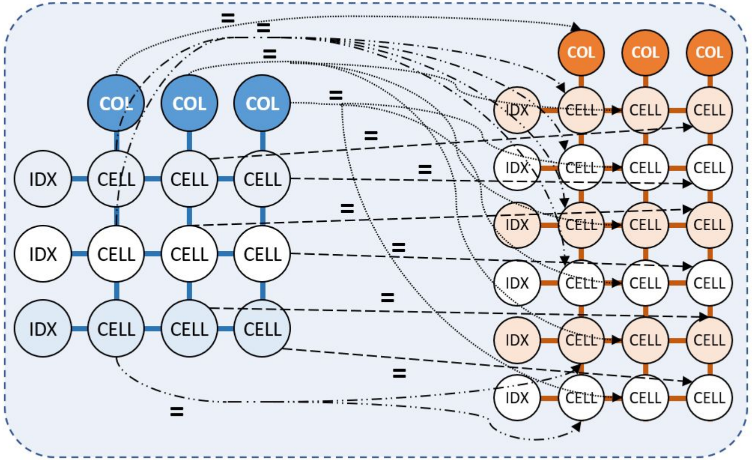
$\subseteq$



Graph Abstraction **of P**

How to Compute Graph Abstractions?

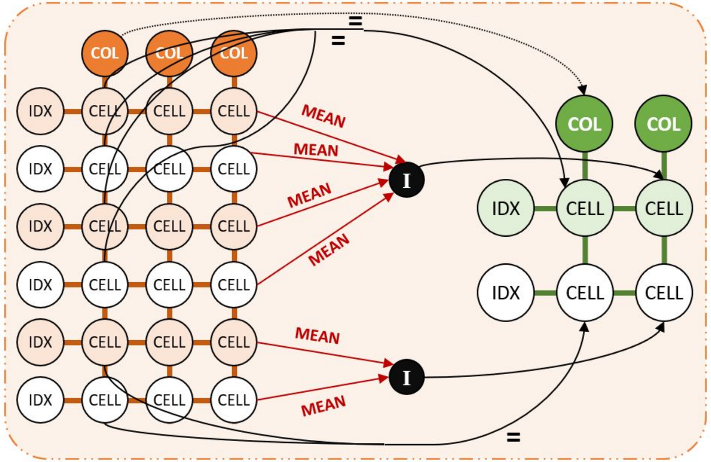
We *augment generators* to create the graph abstraction for a single function call given its arguments



t_1 = `gather(`i`, “Low”, “High”, -“Type”)`

How to Compute Graph Abstractions?

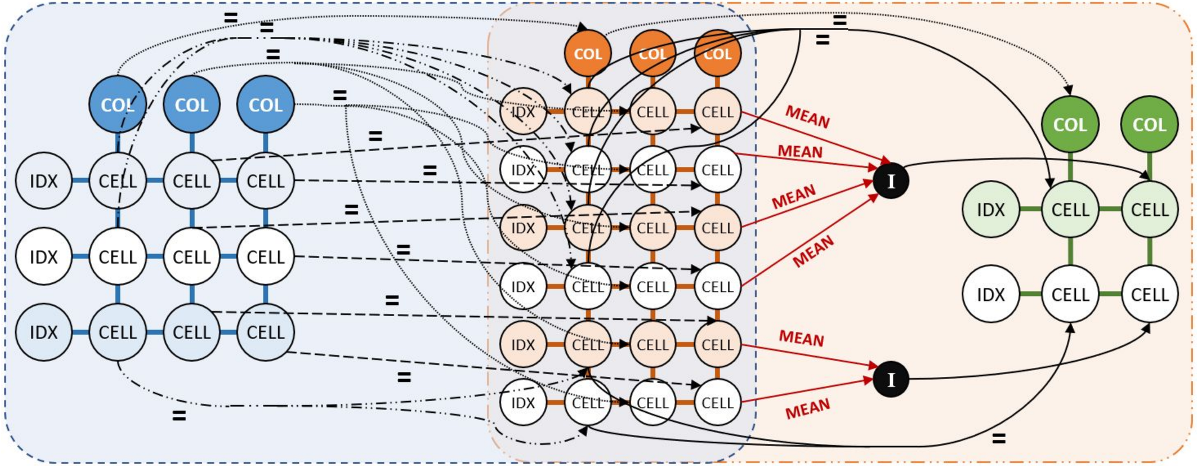
We *augment generators* to create the graph abstraction for a single function call given its arguments



o = group_by(t_1, by="Type", Avg=mean("Value"))

How to Compute Graph Abstractions?

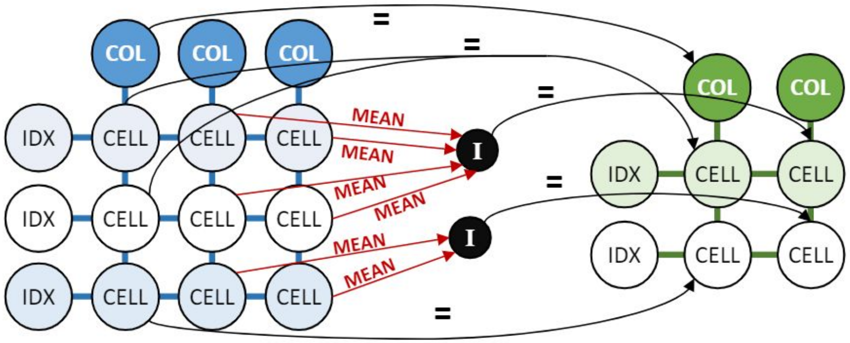
Combine the individual graphs to obtain the graph abstraction



Relationship Propagation Rules

How to Compute Graph Abstractions?

Combine the individual graphs to obtain the graph abstraction



Gauss at a Glance

Capturing Partial Examples and User Intent as a Graph

Gauss Algorithm

Graph-Based Inductive Reasoning to Search Faster

Evaluation

See main talk here - https://youtu.be/M_qGgRR0Y3U

Gauss at a Glance

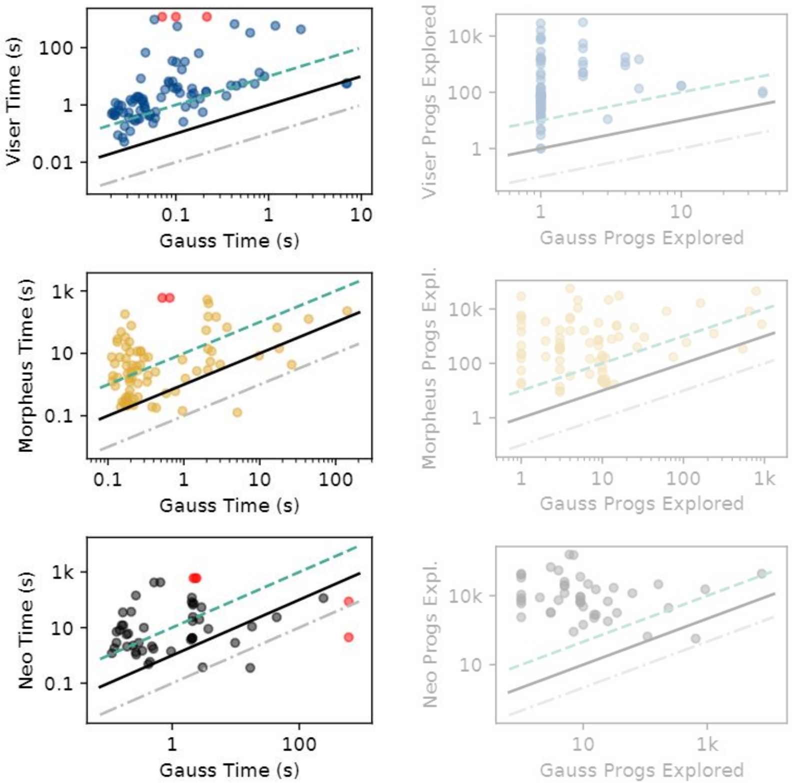
Capturing Partial Examples and User Intent as a Graph

Gauss Algorithm

Graph-Based Inductive Reasoning to Search Faster

Evaluation

What is the Upper-Limit on the Pruning Power of Graph-based Reasoning?



Above ——— : Gauss is Faster

Above – – – : Gauss is 10× Faster

Gauss is faster on average by:

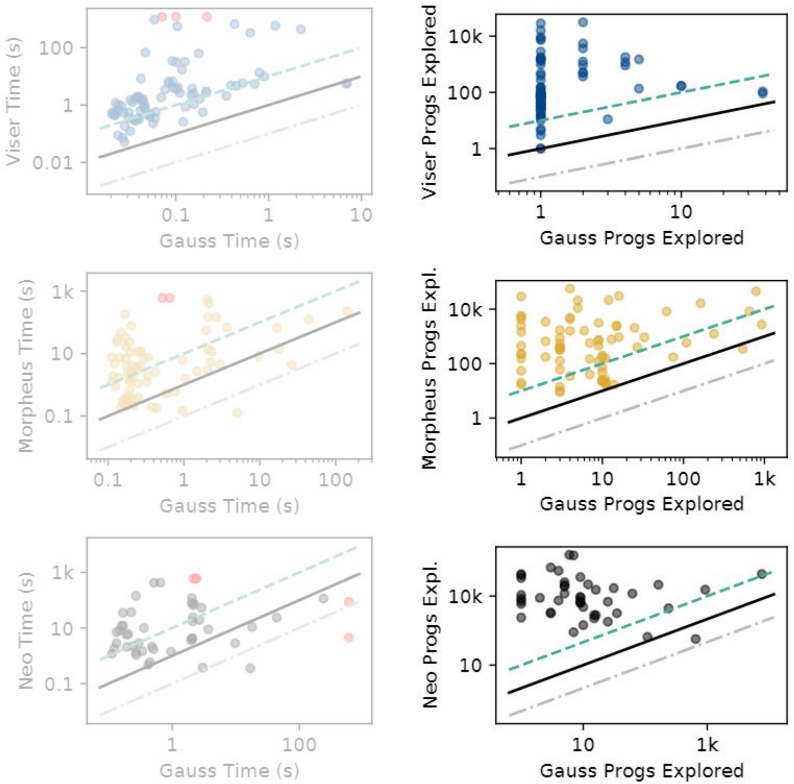
7× vs. Morpheus

26× vs. Viser

7× vs. Neo

Because of additional graph information

What is the Upper-Limit on the Pruning Power of Graph-based Reasoning?



Above : Gauss explores Fewer Progs

Above : Gauss explores 10× Fewer Progs

Gauss searches fewer programs on average by:

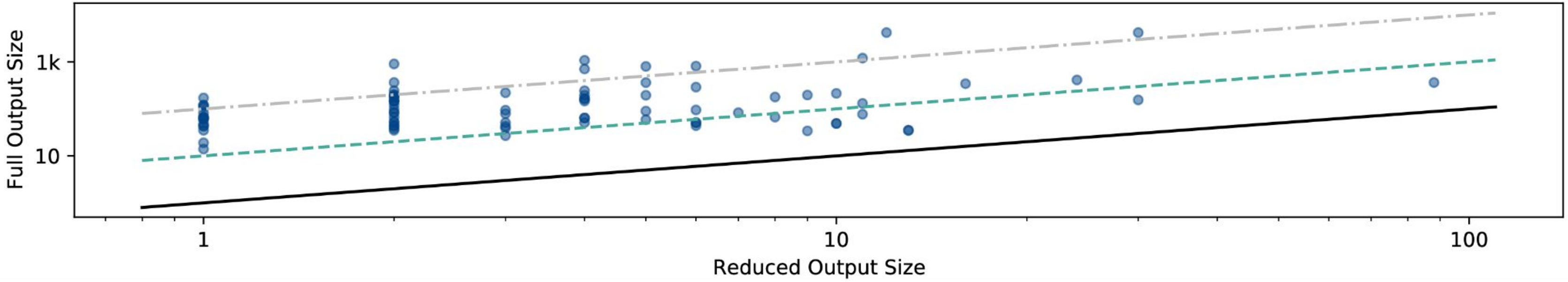
56× vs. Morpheus

73× vs. Viser

774× vs. Neo

Because of additional graph information

Can User Intent Graphs Reduce the Size of Output Specifications?



Above – – – : More than `10×` Reduction
Above – · – · – : More than `100×` Reduction

We find that the maximal reduction of output size—while retaining Gauss's ability to find the solution—is **33×** on average

Informal User Study

- We recruited two graduate students with beginner-level proficiency in Pandas/R
- We gave them 10 tasks
  - 5 to teach them how to work with Gauss
  - And 5 to solve however they wanted (Gauss, Web-Search etc.)
- One solved 10/10 while the other solved 8/10
- Both preferred using Gauss

What about Visualizations?

What's Different?

Hard to Provide **Checkable** Specifications

- How to check a generated program / visualization against the user's intent?
  - Users cannot provide the output viz. - defeats the purpose!
  - Users may not know beforehand what they want
    - They may want to explore and pick what they like
    - They may have a concept in mind (visualize correlations) but not the exact kind of viz. (say heatmap)
  - Partial viz. outputs explored in [1], but limited in scope and types of visualizations produced
    - Can be error-prone since the portion provided must be *exactly* right

[1] Visualization by Example, Wang et al., POPL 2020

What's Different?

How to Define the Search Space?

- Restricting the types/variations of visualizations using DSLs/grammars/generators may be unwise
  - Creating visualizations often involves data preprocessing/shaping operations
  - Need to account for color variations, labeling, legend placement and other stylistic variations?
    - APIs for these (matplotlib, seaborn) are even more unstructured than Pandas!

Research

AutoPandas: Neural-Backed Generators for Program Synthesis
Rohan Bavishi, *Caroline Lemieux, Roy Fox, Koushik Sen, Ion Stoica*
OOPSLA 2019

Gauss

Gauss: Program Synthesis by Reasoning Over Graphs
Rohan Bavishi, *Caroline Lemieux, Koushik Sen, Ion Stoica*
OOPSLA 2021

VizSmith

VizSmith: Automated Visualization Synthesis by Mining Data-Science Notebooks
Rohan Bavishi, *Shadaj Laddad, Hiroaki Yoshida, Mukul R. Prasad, Koushik Sen*
ASE 2021

Tackling Visualizations

- We accept “weak” natural language descriptions of visualizations
- Users can view and select amongst multiple returned visualizations

Specification Modality	Formal Specs	Program Sketches	Traces/ Demonstrations	I/O Examples	Natural Language
Search Space	Sketches and Holes (Partial Programs with Holes)	DSLs / Grammars	Generators	Anything (Arbitrary Text)	
Search Algorithm	Constraint-Solving, CEGIS	Deduction	Enumerative	Stochastic/ ML-Driven	
	Commonly Requires Translation to SAT/SMT	Opt-1: Program-Space Opt-2: Data-Space	w/ Pruning Strategy w/ ML Bias		

Tackling Visualizations

We introduce another search space option - **crowdsourced**
We mine visualization code from Kaggle, templatize it for use in synthesis

Specification Modality	Formal Specs	Program Sketches	Traces/ Demonstrations	I/O Examples	Natural Language
Search Space	Sketches and Holes (Partial Programs with Holes)	DSLs / Grammars	Generators	Crowdsourced (Mined)	Anything (Arbitrary Text)
Search Algorithm	Constraint-Solving, CEGIS	Deduction	Enumerative	Stochastic/ ML-Driven	

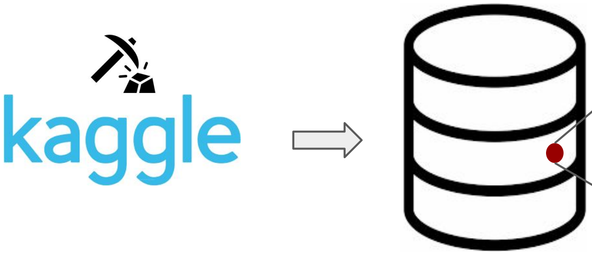
Commonly Requires Translation to SAT/SMT

Opt-1: Program-Space
Opt-2: Data-Space

w/ Pruning Strategy
w/ ML Bias

High-level Overview of VizSmith

Offline Phase



Database of Mined Visualization Functions

Function used to produce Alice’s visualization

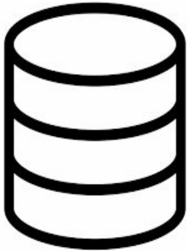
```
def visualization(df, col1, col2):
    import pandas as pd
    import matplotlib.pyplot as plt
    import seaborn as sns
    sns.set()
    tab = pd.crosstab(df0[col1], df0[col2])
    tab.div(tab.sum(1).astype(float), axis=0)\
        .plot(kind="bar", stacked=True)
    plt.ylabel('Text-1’)
    plt.title('Text-2’)
```

VizSmith mines Kaggle to create a database of visualization functions

High-level Overview of VizSmith

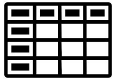
Offline Phase

kaggle



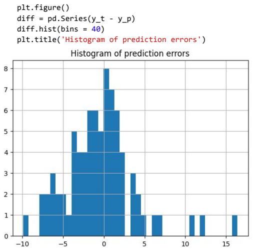
Database of Mined
Visualization Functions

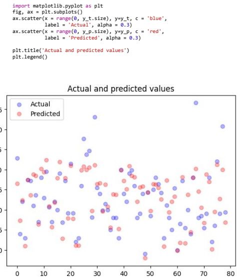
Online Phase



Columns
to
Visualize

Text
Query





VizSmith matches function metadata against the queries and returns generated viz.

Demo

https://github.com/rbavishi/vizsmith-demo

Video Link: https://www.youtube.com/watch?v=GWROeYA_l-U

Technique

See main talk here - https://youtu.be/eJCS_PfpuaM

Evaluation

The Mining Process

- We mined ~3k notebooks across 10 popular competitions on Kaggle
- We obtained ~9k functions in total, with ~7k passing the reusability check

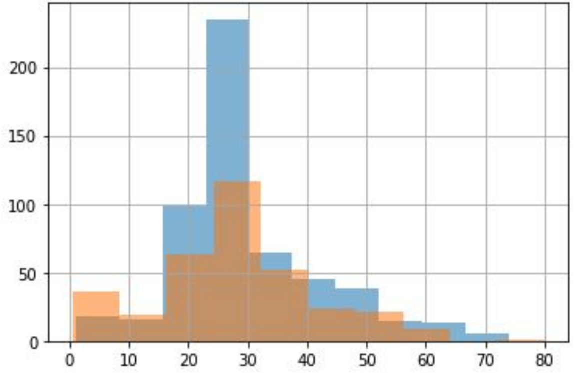
What is the synthesis performance?

How do we get benchmarks?

We create benchmarks automatically as follows:

Dataframe Input: `df_train`
Visualized Cols: `['Survived', 'Age']`
Text query: `Distribution of Age by Survived`

In [21]:
```
df_train.groupby('Survived').Age.hist(alpha=0.5)
```



Distribution of age by Survived

What is the synthesis performance?

- Cross-Project Setup: For a benchmark, solve it using visualization functions mined from **other** competitions
- We measure top-10 accuracy
    - Baseline: VizSmith without reusability analysis

Exact Accuracy = 5%

Low because stylistic variations hard to match with NL

What is the synthesis performance?

- Cross-Project Setup: For a benchmark, solve it using visualization functions mined from **other** competitions
- We measure top-10 accuracy
  - Baseline: VizSmith without reusability analysis
- We sampled 50 benchmarks and analyzed results manually, accounting for stylistic variations

Accuracy = 56%
(Baseline = 46%)

VizSmith **explored 50% less** functions than the baseline

Limitations / Future Work

- VizSmith's retrieval relies heavily on the natural language found in notebooks
    - They are often of poor quality and imprecise / irrelevant
    - We are exploring the use of auto-documentation techniques to improve the indexing

- We have not performed a formal user study and hence end-to-end performance may not reflect real-world usage
    - We have released the tool and examples of different kinds of plots that can be synthesized by VizSmith at https://github.com/rbavishi/vizsmith-demo

Summary

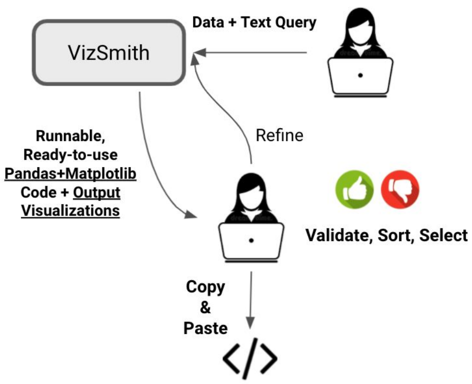
Procedural Viz. Authoring Tools can be hard to learn and cumbersome to use

matplotlib

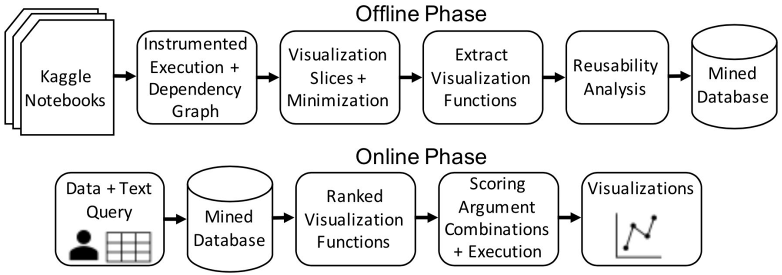
seaborn

stackoverflow

VizSmith accepts data and a text query and returns ready-to-use visualizations



Offline, VizSmith mines Kaggle to build a database of visualization functions. It uses metamorphic testing to assess quality.
In an online phase, it uses TF-IDF to match against functions and return visualizations



- VizSmith's database exercises 289 API functions across 12 libraries, covering a variety of data transformation, plotting and styling APIs
- In an end-to-end setting VizSmith achieves **56%** top-10 accuracy on synthetic benchmarks. Notably, its reusability analysis improves top-10 accuracy by **10%** and reduces the search space by **50%**

What’s Next?

What's Next?

Codex/GPT-3 etc. have been shown to be surprisingly good at natural-language based program synthesis

GitHub Copilot — A New Generation of AI Programmers

GitHub, Microsoft, and OpenAI have reached a new milestone.

Alberto Romero Jul 1,

The New York Times

Meet GPT-3. It Has Learned to Code (and Blog and Argue).

The latest natural-language system generates tweets, pens poetry, summarizes emails, answers trivia questions, translates languages and even writes its own computer programs.

Program Synthesis with Large Language Models

Jacob Austin* **Augustus Odena***

Maxwell Nye† **Maarten Bosma** **Henryk Michalewski** **David Dohan** **Ellen Jiang** **Carrie Cai**

Michael Terry **Quoc Le** **Charles Sutton**

Traces/ Demonstrations

I/O Examples

Natural Language

Search Space

Generators

Crowdsourced (Mined)

Anything (Arbitrary Text)

Search Algorithm

Stochastic/ ML-Driven

Translation to SAT/SMT

Opt-2: Data-Space

w/ ML Bias

Promises and Pitfalls

The Promise:
Large language models have the potential to greatly amplify the benefits of a **multi-modal** approach that includes **natural language**

The Pitfalls:
They are **error-prone**. Models do not understand semantics. They can mistype variables, forget imports, use incorrect functions, arguments etc.

The Opportunity

Repair Output of LMs

OR

Constrain the Output of LMs

- Explore PL techniques to automatically repair the output of LMs or constrain the output to ensure correctness w.r.t. some syntactic/semantic criteria
- Example: [1] explores post-processing techniques to correct the output of LMs for Pandas using input-output examples

[1] Jigsaw: Large Language Models meet Program Synthesis, Jain et al. ICSE 2022

The Natural Language Specification / Prompt

Models are **sensitive** to the way you describe a task

A beginner may have to try multiple queries/prompts to get the model to do it right

Writing Natural Language Can Be Non-Trivial

Prompt:

```
Description: Plot pie-chart of the "Category1" column in "df"

Python Code:
df.plot.pie(y="Category1")

Description: Visualize the correlation matrix of "df" as a heatmap using seaborn.

Python Code:
```

Few-Shot Learning
(only 1 example shown)

Writing Natural Language Can Be Non-Trivial

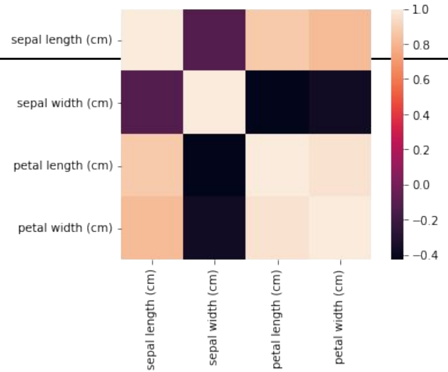
Prompt:

```
Description: Plot pie-chart of the "Category1" column in "df"

Python Code:
df.plot.pie(y="Category1")

Description: Visualize the correlation matrix of "df" as a heatmap using seaborn.

Python Code:
```

Few-Shot Learning
(only 1 example shown)

Completion:

```
import seaborn as sns
sns.heatmap(df.corr(), square=True)
```



Writing Natural Language Can Be Non-Trivial

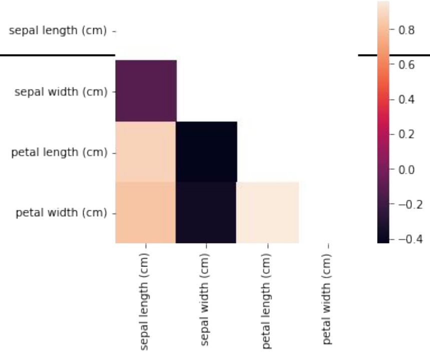
Prompt:

```
Description: Plot pie-chart of the "Category1" column in "df"

Python Code:
df.plot.pie(y="Category1")

Description: Visualize the correlation matrix of "df" as a heatmap using seaborn.

Python Code:
```

Few-Shot Learning
(only 1 example shown)

Completion:

```
import seaborn as sns
sns.heatmap(df.corr(), square=True)
```



Let's say we just want the lower-triangle (remove symmetry)

Writing Natural Language Can Be Non-Trivial

Prompt:

```
Description: Plot pie-chart of the "Category1" column in "df"

Python Code:
df.plot.pie(y="Category1")

Description: Visualize the lower triangle of the correlation matrix of "df" as a
heatmap using seaborn.

Python Code:
```

Few-Shot Learning
(only 1 example shown)

Completion:

```
import numpy as np
sns.heatmap(df.corr(), mask=np.zeros_like(df.corr(), dtype=np.bool),
cmap=sns.diverging_palette(220, 10, as_cmap=True), square=True)
```

Wrong!

Writing Natural Language Can Be Non-Trivial

Prompt:

```
Description: Plot pie-chart of the "Category1" column in "df"

Python Code:
df.plot.pie(y="Category1")

Description: Visualize the correlation matrix of "df" as a heatmap using seaborn
and remove the upper triangle

Python Code:
```

Few-Shot Learning
(only 1 example shown)

Completion:

```
import numpy as np
sns.heatmap(df.corr(), mask=np.zeros_like(df.corr(), dtype=np.bool),
cmap=sns.diverging_palette(220, 10, as_cmap=True), square=True)
```

No change!

Writing Natural Language Can Be Non-Trivial

Prompt:

```
Description: Plot pie-chart of the "Category1" column in "df"

Python Code:
df.plot.pie(y="Category1")

Description: Visualize the correlation matrix of "df" as a heatmap using seaborn
and mask the upper triangle

Python Code:
```

Few-Shot Learning
(only 1 example shown)

Completion:

```
import numpy as np
sns.heatmap(df.corr(), mask=np.zeros_like(df.corr(), dtype=np.bool),
cmap=sns.diverging_palette(220, 10, as_cmap=True), square=True)
```

No change!

Writing Natural Language Can Be Non-Trivial

Prompt:

```
Description: Plot pie-chart of the "Category1" column in "df"

Python Code:
df.plot.pie(y="Category1")

Description: Visualize the correlation matrix of "df" as a heatmap using seaborn
and masking the upper triangle

Python Code:
```

Few-Shot Learning
(only 1 example shown)

Completion:

```
import seaborn as sns
sns.heatmap(df.corr(), mask=np.triu(df.corr(), k=1), cmap="RdBu_r")
```

The Opportunity

Can we guide users when writing NL?

Can we help users explore / discover feasible NL queries?

[1] Jigsaw: Large Language Models meet Program Synthesis, Jain et al. ICSE 2022

An Idea: Utilize Code-to-NL Capabilities of LMs

LMs like Codex are good at describing code!

Model *completes* the prompt:

* Plot a bar-plot of "col0" in the descending order with index as the x-axis.
* Use seaborn

Few-Shot Learning Examples

```
----
Python Code:
def visualization(df0):
    import seaborn as sns
    sns.pairplot(df0)

Description:
* Plot all the pairwise relationships
----
Python Code:
def visualization(df0, col0, col1):
    import seaborn as sns
    gb = df0.groupby(col0)[col1].value_counts().to_frame().rename({col1: 'Text-1'}, axis = 1).reset_index()
    sns.barplot(x = col0, y = 'Text-1', data = gb, hue = col1, palette = sns.color_palette("hls", 8))

Description:
* Plot counts of unique values in "col0".
* Add "Text-1" as the y-label.
* Use "hls" as the colorscheme.
----
Python Code:
def visualization(df0, col0):
    import seaborn as sns
    df0.sort_values(by=[col0], inplace=True, ascending=False)
    sns.barplot(df0[col0].index, df0[col0])

Description:
```

Code to describe

Prompt provided to Codex

An Idea: Utilize Code-to-NL Capabilities of LMs

LMs like Co
descr

We are working on combining this capability with our mining approach à la VizSmith

- Mined snippets represent what users write, and their descriptions can serve as suggestions for the end-user!

Model *completes* the prompt:

* Plot a bar-plot of "col0" in the descending order with index as the x-axis.
* Use seaborn

Few-Shot Learning Examples

```
----
Python Code:
def visualization(df0):
    import seaborn as sns
...ext-1'}, axis = 1).reset_index()
...or_palette("hls", 8))

Description:
* Plot counts of unique values in "col0".
* Add "Text-1" as the y-label.
* Use "hls" as the colorscheme.
----
Python Code:
def visualization(df0, col0):
    import seaborn as sns
    df0.sort_values(by=[col0], inplace=True, ascending=False)
    sns.barplot(df0[col0].index, df0[col0])

Description:
```

Code to describe

Prompt provided to Codex

Thank you to my Collaborators!

Caroline Lemieux
MSR, UBC

Roy Fox
UC Irvine

Koushik Sen
UC Berkeley

Ion Stoica
UC Berkeley

Shadaj Laddad
UC Berkeley

Hiroaki Yoshida
Fujitsu

Mukul Prasad
Fujitsu