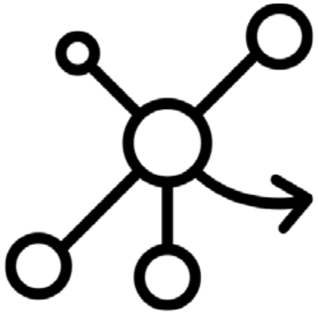
TEL AVIV UNIVERSITY אוניברסיטת תל אביב

GUIDED **DATA SCIENCE**

Project Number: 18-1-1-1638

Project Presentation



DATA 'SCIENCE'

Vaguely: **Data Science** is an interdisciplinary field about scientific methods, processes and systems to extract **knowledge** or **insights** from data.

glassdoor 50 Best Jobs in America for 2019

	Job Title	Median Base Salary	Job Satisfaction	Job Openings
#1	Data Scientist	$108,000	4.3/5	6,510

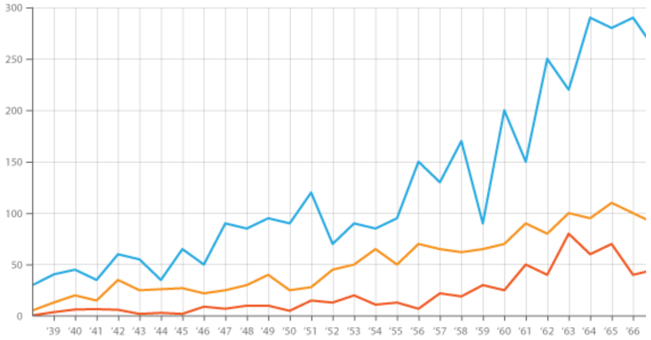
WHAT DO **DATA SCIENTISTS** DO

DATA SCIENTIST

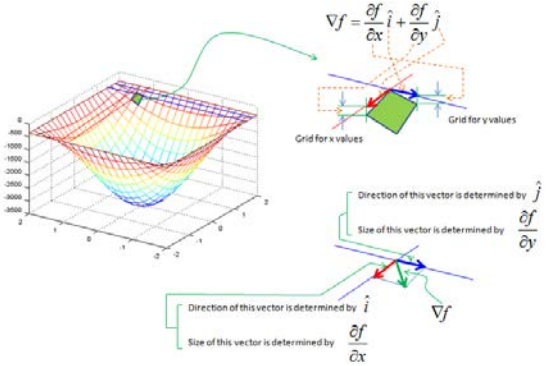
What my friends think I do

What my mom think I do

What society thinks I do



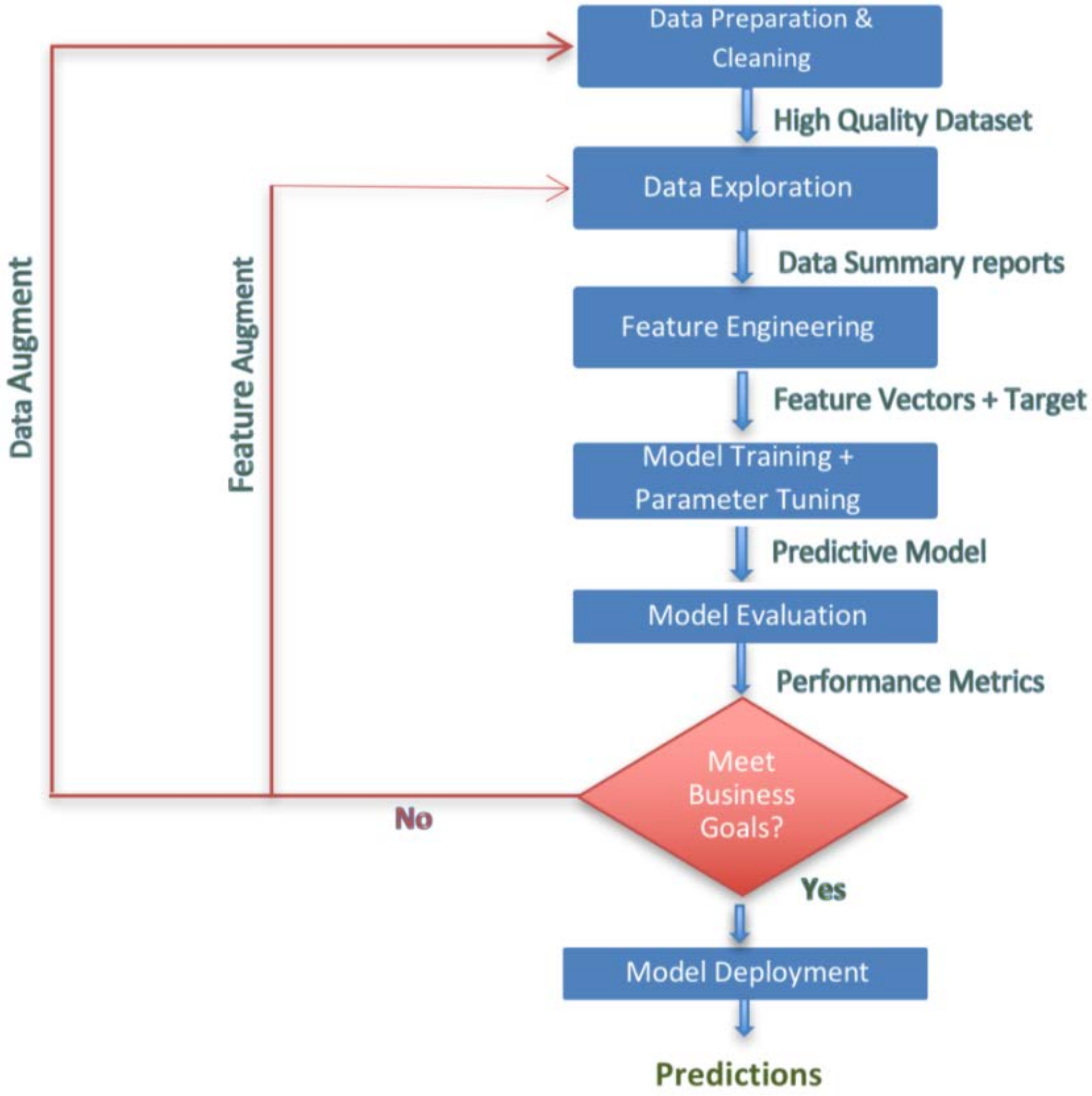
What my boss thinks I do



What I think I do

What I actually do

WHAT DO **DATA SCIENTISTS *REALLY*** DO



IS IT **DIFFICULT?**

YES.

1. **"Big Data"**: Volume, Velocity, Variety, but also: Veracity (accuracy) and Volatility

2. **"Poor Data"**: Dirty data, missing values

3. **"Domain understanding"**: lack of understanding of the investigated domain may lead to poor results, wrong/irrelevant insights

4. **"Lack of 'Know-hows'"**: Best or common-practices are not easily accessible or shared

HOW CAN WE **HELP?**

Existing efforts:

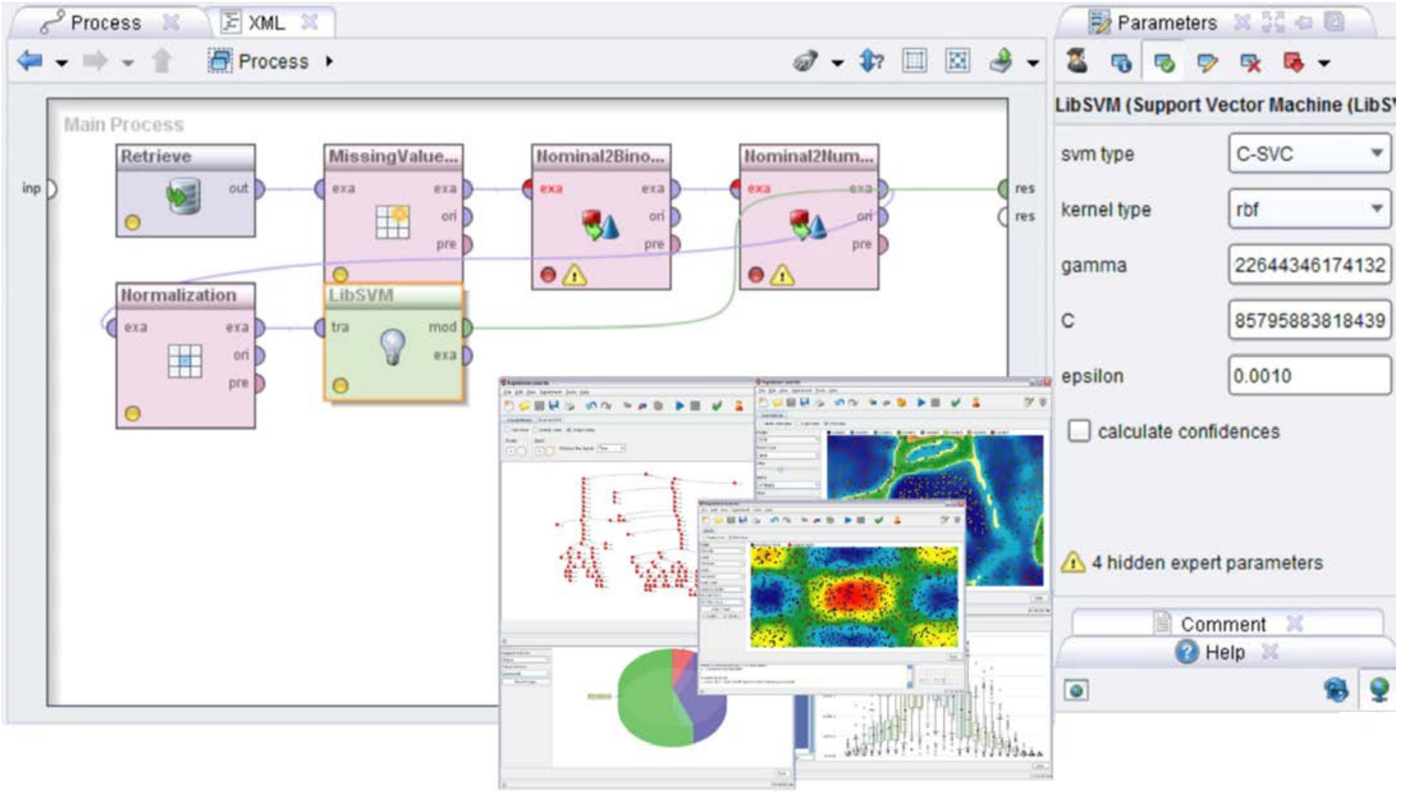
1. Big Data Processing:

2. Software platforms:

HOW CAN WE **HELP?**

Existing efforts:

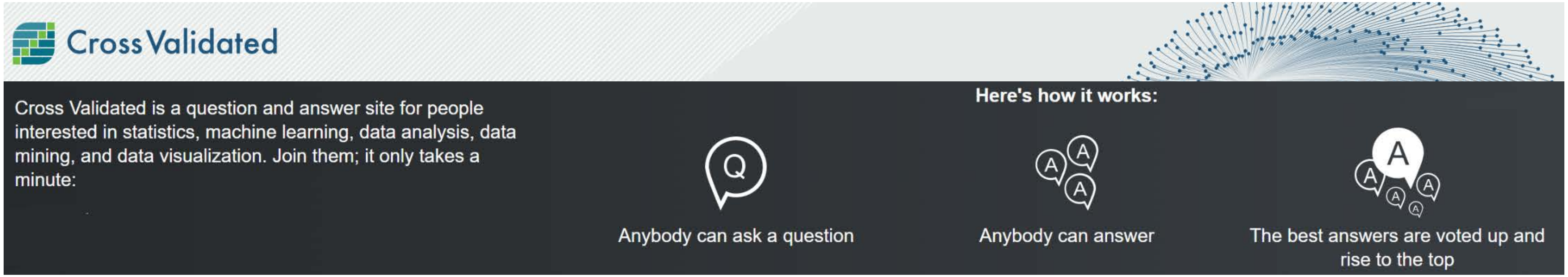
3. **Tools for non-programmers**



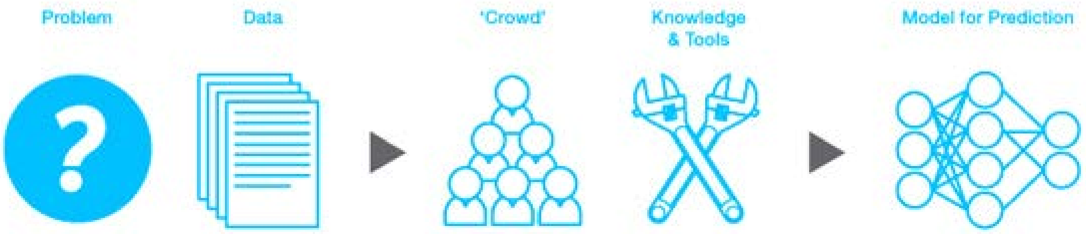
HOW CAN WE **HELP?**

Existing efforts:

4. **Knowledge sharing:**



kaggle



HOW CAN ***WE* HELP?**

Consider the following scenario:

1. Jennifer, a new data scientist is recruited to a data science team in a firm.

2. She is assigned her first task, predict the sales revenue from a specific product line, in Q4 2019.

3. First, she asks other people on the team for any previous work, and is sent a mixed bag of presentations, emails, and documents.

However… **The previous work doesn't have the up-to-date code, or is not adequate for her assignment**

4. Jennifer tracks down the local copy on the original author's machine or an outdated GitHub link.

5. After fiddling with the code, Jennifer realizes it's slightly different from what made the plots she received in the email…

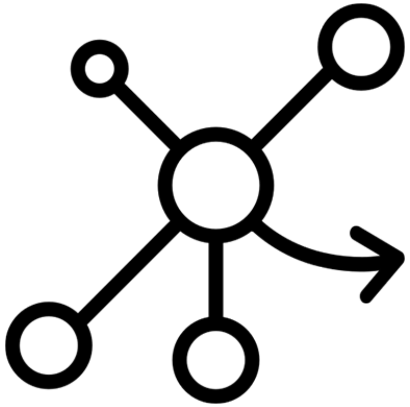
… **After spending time adapting previous work,**

she gives up and start from scratch

WHAT DO ***WE* WANT?**

A recommender system for data scientists that will:

1. Hook in the data science UI
2. “Understand” the dataset and task
3. Leverage other people’s relevant knowledge
4. Automatically adapt it to fit in the current user’s context.
5. Suggest adequate, context-sensitive, code-snippets.

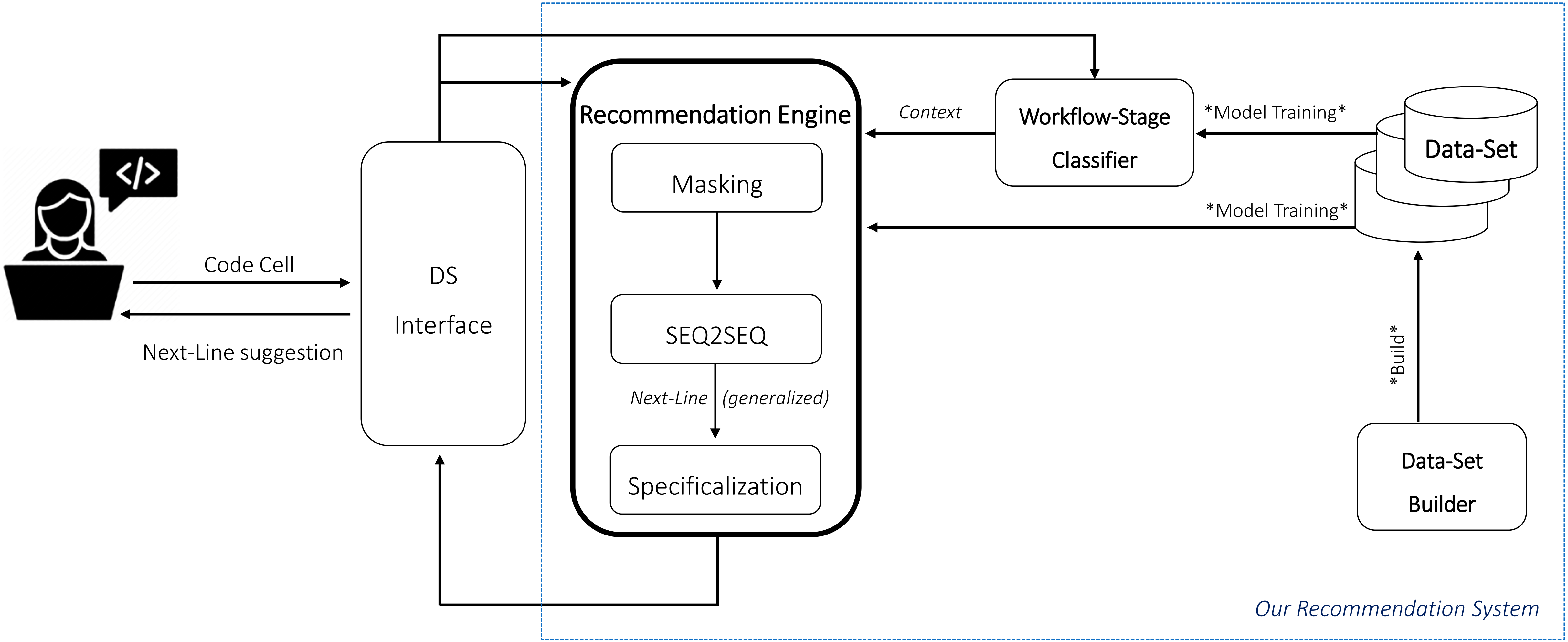


WHAT DO ***WE* NEED?**

To design such a framework, we need:

1. A repository of reproducible data science **code-snippets**.
2. A method for analyzing the user's **context.**
3. A similarity notion and an efficient algorithm for **identifying similar code-snippets**
4. A procedure that takes a code-snippet and **adapt it to the user's context**

SYSTEM **ARCHITECHTURE**



STEP1: **DATASET BUILDER**

create a repository of reproducible data science **code-snippets**

DS **NOTEBOOKS**

jupyter notebooks (Python), are web based interactive scripting environments.

They are extremely popular among data scientists, as they tie together both the code snippets and the results.

And even better- they are **reproducible**!

Import Python Libraries

```
In [1]: import pandas as pd
        import numpy as np
        import seaborn as sns
        import sklearn
        import matplotlib.pyplot as plt
```

Reading the Income Classification Dataset

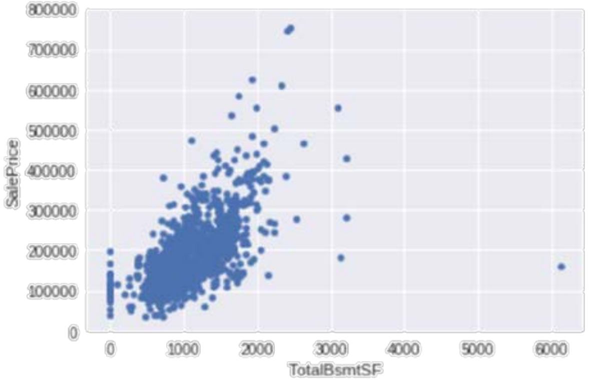
```
In [2]: train_data = pd.read_csv('adult.data', header=None)
```

```
In [3]: test_data = pd.read_csv('adult.test')
        test_data.head(10)
```

Out[3]:

														|1x3 Cross validator
25	Private	226802	11th	7	Never-married	Machine-op-inspct	Own-child	Black	Male	0	0	40	United-States	<=50K
38	Private	89814	HS-grad	9	Married-civ-spouse	Farming-fishing	Husband	White	Male	0	0	50	United-States	<=50K
28	Local-gov	336951	Assoc-acdm	12	Married-civ-spouse	Protective-serv	Husband	White	Male	0	0	40	United-States	>50K
44	Private	160323	Some-college	10	Married-civ-spouse	Machine-op-inspct	Husband	Black	Male	7688	0	40	United-States	>50K

```
In [8]: #scatter plot totalbsmtsf/saleprice
        var = 'TotalBsmtSF'
        data = pd.concat([df_train['SalePrice'], df_train[var]], axis=1)
        data.plot.scatter(x=var, y='SalePrice', ylim=(0,800000));
```

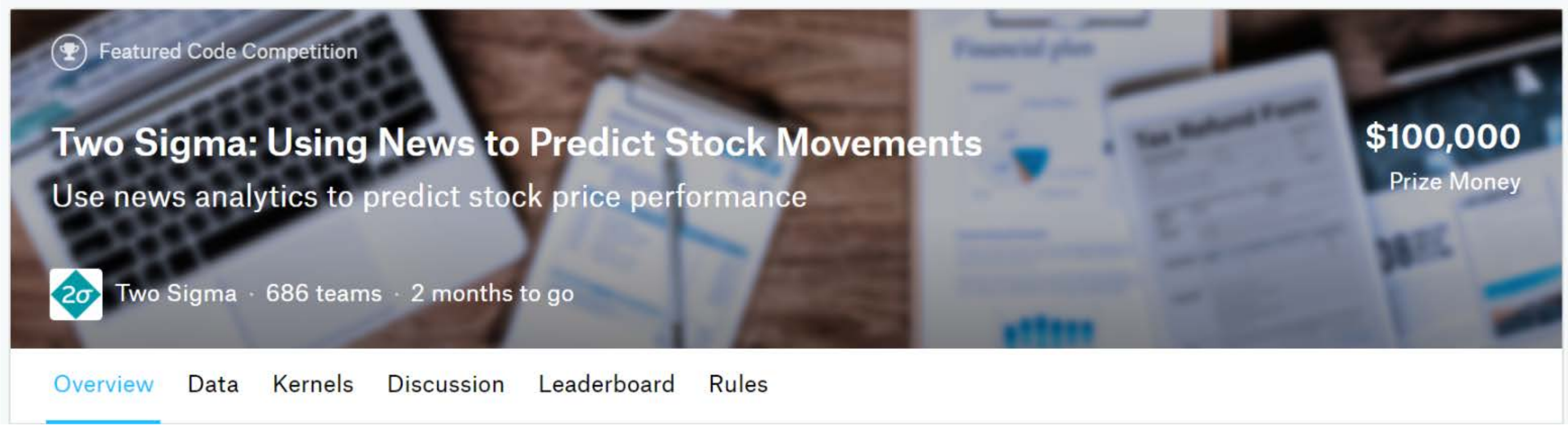


DS **NOTEBOOKS**

kaggle (acquired by Google) is a DS knowledge sharing platform supporting:

1. Hosting analytics competitions
2. Publishing data sets and notebooks

== A publicly available source for {datasets}x{notebooks}



DATASET **BUILDER**

- Automated web crawler that gets tag words as input and downloads all of the relevant Jupyter notebooks and all available metadata from Kaggle
- Parses the downloaded data into a csv file:

Cell_id	Source	Outputs	Execution_count	Notebook	Dataset_name
Unique key	Cell source code	After Execution	Execution order	Notebook name	Dataset name
…	…	…	…	…	…

* AST and Masked representation of the code cells are computed later for each cell's source code

- Built using Selenium and Kaggle API

OUR **DATASET**

- 146 Datasets (80 of them for competitions)
- 19,081 Jupyter notebooks (avg. of ~130 notebooks per dataset)
- 296,281 Cells of code (avg. of ~16 cells per notebook)
- Avg. of ~7 lines per cell

OUR **DATASET**

- 12,693 empty cells ☹
- 34,810 useless cells (only contain prints or comments) ☹
- A lot of similar notebooks for each dataset

STEP2: **Workflow Stage Classifier**

Analyze the **context** of the user's code

DATA SCIENCE **WORKFLOW**

- CRISP-DM Data Mining Methodology:
  - Business Understanding:

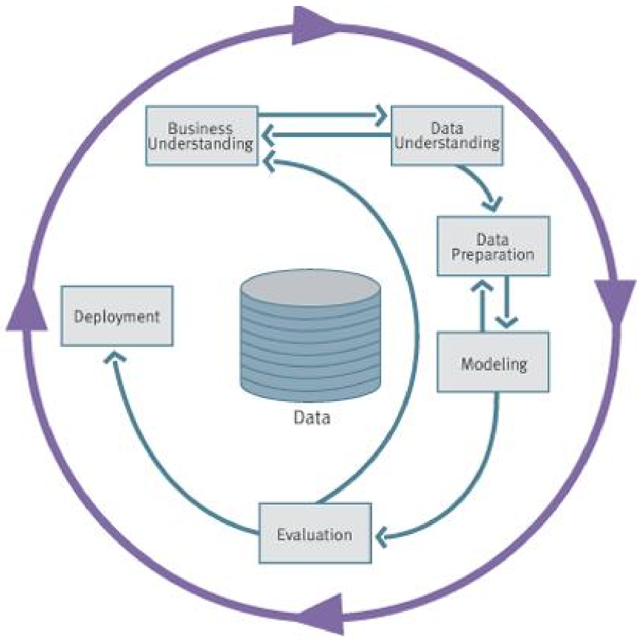
    Why? What? Goal of the project? What has already been done? Competition?
  - Data Understanding:

    What can we achieve? What are the low hanging fruits? How can we address our problem?
    What is the cost/effort of dealing with or getting more data?
  - Data Preparation:

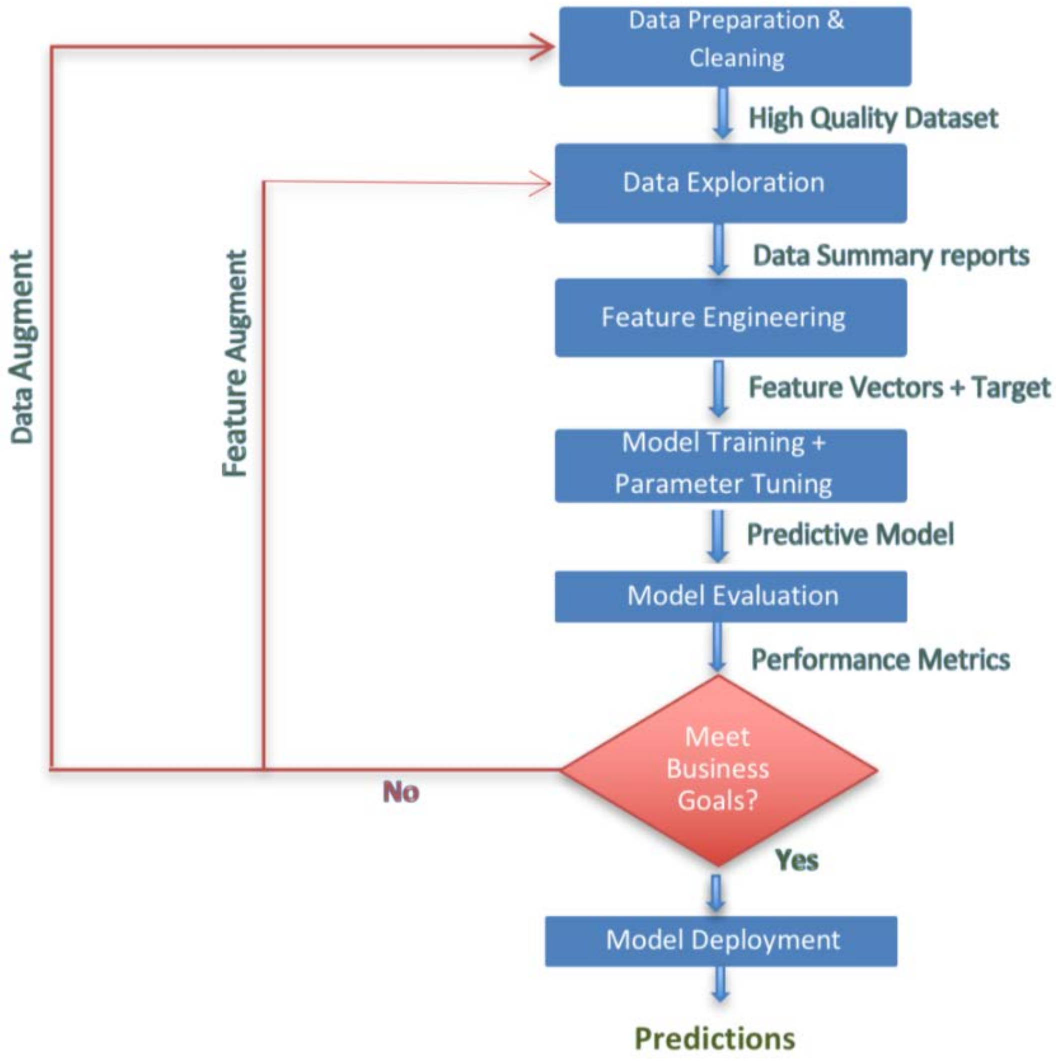
    Data Cleaning (missing values, outliers), Data discretization, Data normalization
    Dimensionality reduction (Feature selection)
  - Modeling:

    Build Model, Train Model, Parameter Tuning
  - Evaluation:

    Evaluate the model's performance, does it meet the goals?



DATA SCIENCE **WORKFLOW**

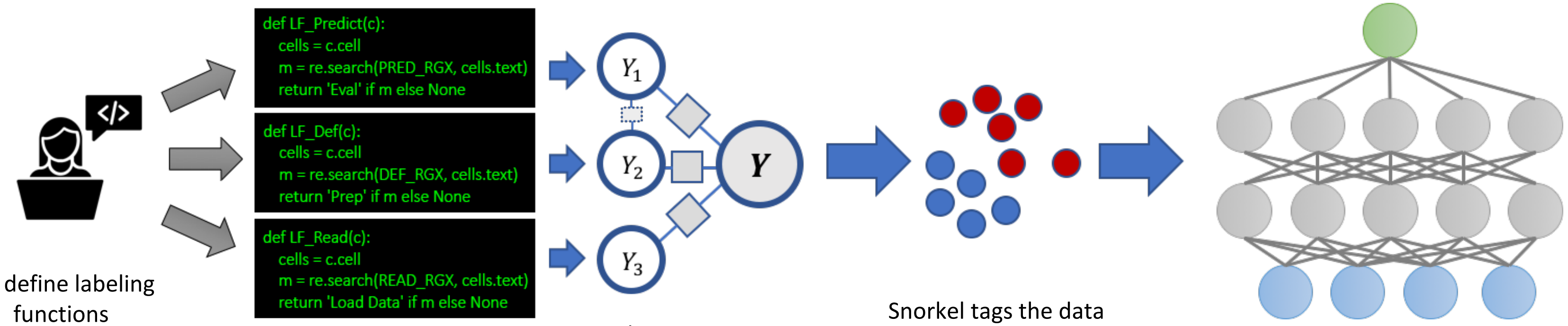


Classes for our classifier:

- Imports
- Load Data
- Data Exploration
- Data Preparation
- Model Training and Parameter Tuning
- Evaluation

SNORKEL **WEAK SUPERVISION**

- Our collected data is unlabeled (there is no known workflow stage)
- We used recurrent functions of each workflow stage to write labeling functions- scripts that programmatically label data.
- The resulting labels are very noisy (collisions and mistakes), but Snorkel automatically models this process- learning, essentially, which labeling functions are more accurate than others (using a small hand labeled data).
- We then use the labeled data generated by the generative model to train our end model



we define labeling functions

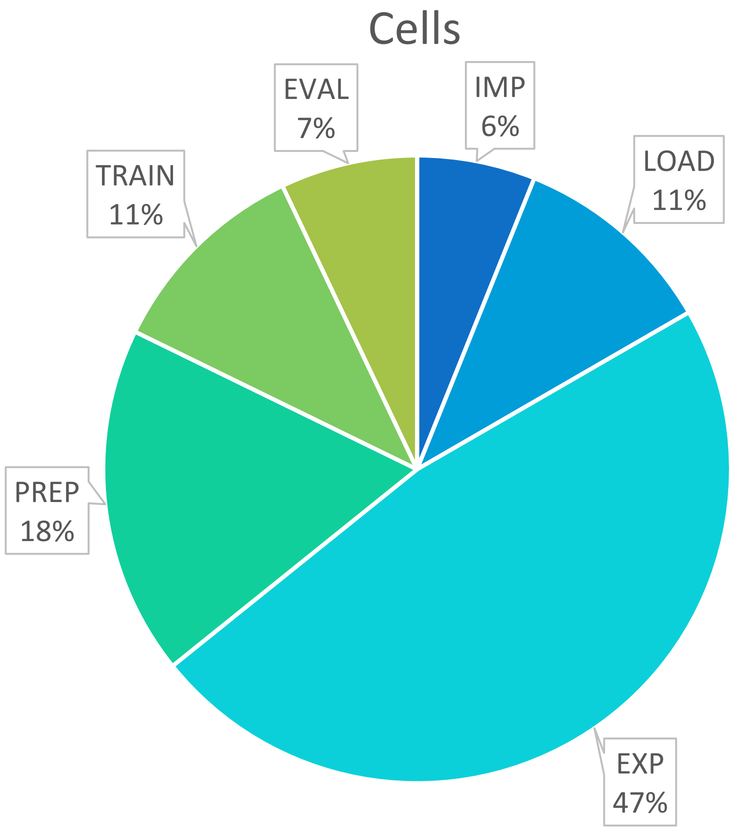
The generative model gives weights (learns the distribution)

Snorkel tags the data – Labeled Data

We use the labeled data to train the end model (LSTM Classifier)

OUR SNORKEL ***LABELED* DATASET**

- Hand-Tagged ~1000 Cells
- ~30 different Labeling Functions



WORKFLOW STAGE **CLASSIFIER**

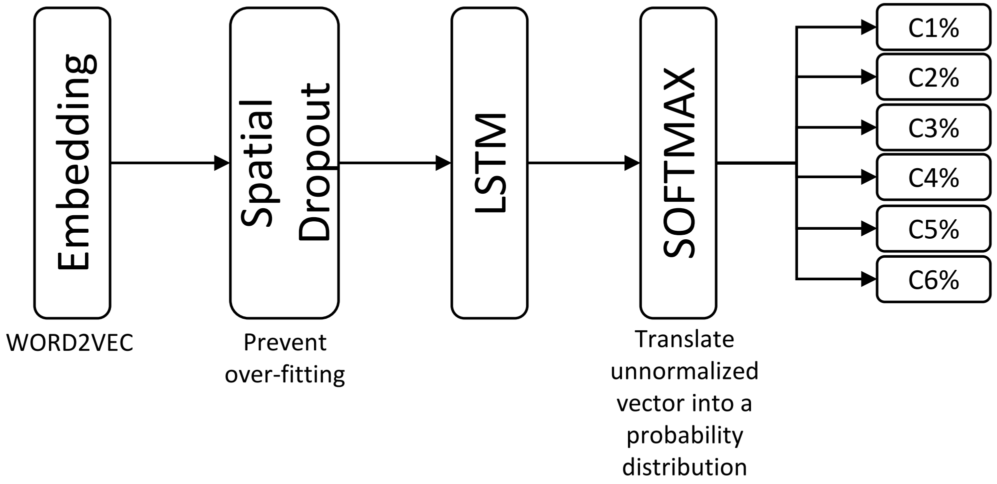
Preprocessing:

- We had imbalanced classes- a lot of "Exploration" (makes sense), We sampled a fixed number of cells from each class.
- Turned to-lower and filtered special chars and comments
- Keras Tokenizer turned all used words to a vocabulary. Found ~50,000 unique tokens.
- Only most common 8,000 words were kept
- Translated each cell to a sequence of integers, padded to a fixed max length, WORD2VEC Embedding

WORKFLOW STAGE **CLASSIFIER**

Classifier:

- LSTM model for multi-label classification (Using Keras sequential model)
- Long Short Term Memory, capable of learning long-term dependencies. Since there is a dependency between cells we figured such a model could benefit us.



More on LSTM

WORKFLOW STAGE **CLASSIFIER**

Evaluation:

- Weak Supervision evaluation:

  We check the Categorical Accuracy over the test set – how many cells where tagged right (according to the hand-tagged data). We achieved an accuracy of 82.3%.

- End Model (classifier) evaluation:

  Again, we look at the Categorical Accuracy over the test set (it's a different test set than before of course, we train-test-split the tagged data). We achieved an accuracy of 86.7%.

- Classification reports:

Snorkel

		precision	recall	f1-score	support
Load	1.0	0.71	1.00	0.83	29
Prep	2.0	0.96	0.69	0.80	96
Train	3.0	0.94	0.89	0.92	55
Eval	4.0	0.79	0.70	0.75	44
Exp	5.0	0.72	0.91	0.81	89
Import	6.0	1.00	1.00	1.00	10
	micro avg	0.82	0.82	0.82	323
	macro avg	0.85	0.87	0.85	323
	weighted avg	0.85	0.82	0.82	323

Classifier

		precision	recall	f1-score	support
Load	0	0.94	0.92	0.93	1247
Prep	1	0.82	0.84	0.83	1213
Train	2	0.86	0.89	0.87	1207
Eval	3	0.87	0.82	0.85	1257
Exp	4	0.83	0.83	0.83	1286
Import	5	0.89	0.90	0.89	1246
	micro avg	0.87	0.87	0.87	7456
	macro avg	0.87	0.87	0.87	7456
	weighted avg	0.87	0.87	0.87	7456

WORKFLOW STAGE **CLASSIFIER**

Examples:

```
txt = ["accr = model.evaluate(X_test,y_test) print('Test set\n  Loss: {:0.3f}\n  Accuracy: {:0.3f}'.format(accr[0],accr[1]))"]
seq = tokenizer.texts_to_sequences(txt)
padded = pad_sequences(seq, maxlen=max_len)
pred = model.predict(padded)
print(pred, labels[np.argmax(pred)])
```

```
[[0.00180286 0.00191248 0.04635391 0.94455606 0.00403245 0.00134233]] Eval
```

```
txt = ["import library\nimport otherlibrary"]
seq = tokenizer.texts_to_sequences(txt)
padded = pad_sequences(seq, maxlen=max_len)
pred = model.predict(padded)
print(pred, labels[np.argmax(pred)])
```

```
[[0.02677174 0.01430851 0.03090717 0.06032386 0.00464913 0.86303955]] Import
```

```
txt = ["model = KNeighborsClassifier(n_neighbors=3), model.fit(x, y)"]
seq = tokenizer.texts_to_sequences(txt)
padded = pad_sequences(seq, maxlen=max_len)
pred = model.predict(padded)
labels = ['Load', 'Prep', 'Train', 'Eval', 'Explore', 'Import']
print(pred, labels[np.argmax(pred)])
```

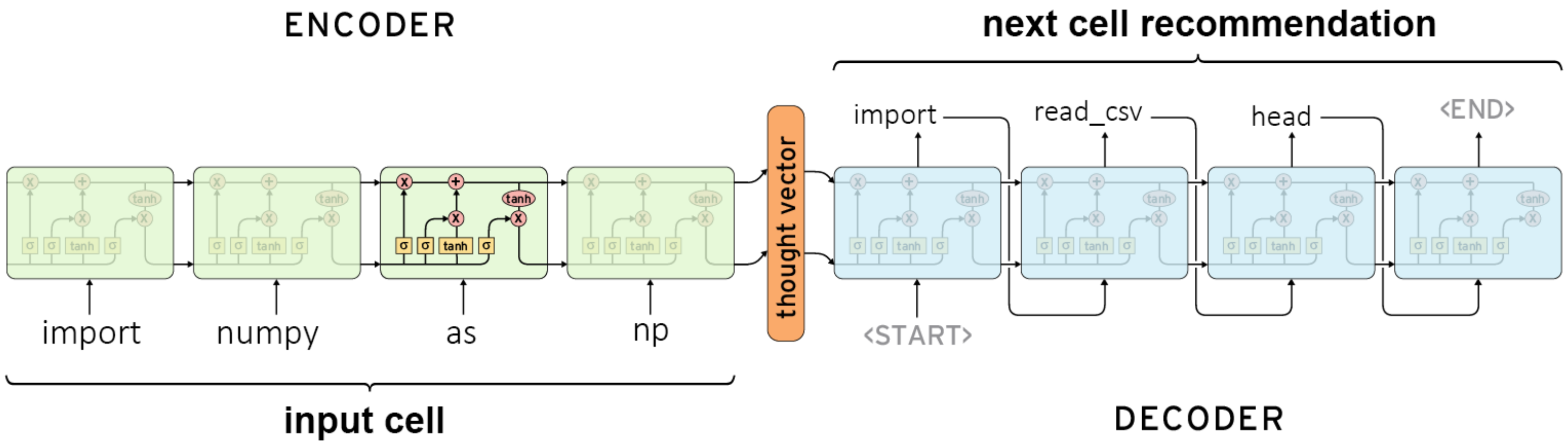
```
[[2.8621647e-04 2.2268973e-03 9.6395564e-01 3.2358591e-02 4.4172912e-04
  7.3095254e-04]] Train
```

STEP3: **Recommendation Engine**

Generate next-step **recommendation**

SEQUENCE-TO-**SEQUENCE**

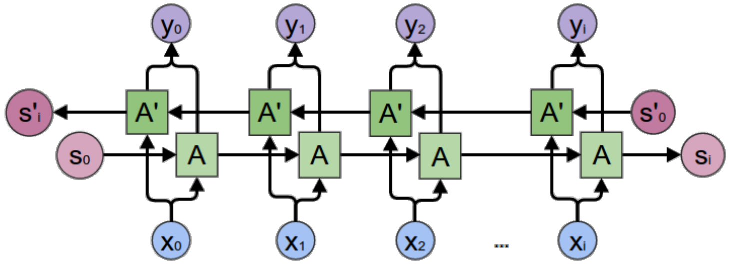
- Conversational models (Chatbots) are a hot topic in artificial intelligence research
- Chatbots can be found in a variety of settings (customer service etc.). These bots are often powered by retrieval-based models, which output predefined responses to questions
- Google's Neural Conversational Model marked a large step towards multi-domain generative conversational models
- Sequence-to-Sequence learning is done with an Encoder-Decoder Framework of RNNs
- We used this to implement a next-line-recommendation chatbot, using PYTORCH



OUR **MODEL**

Encoder

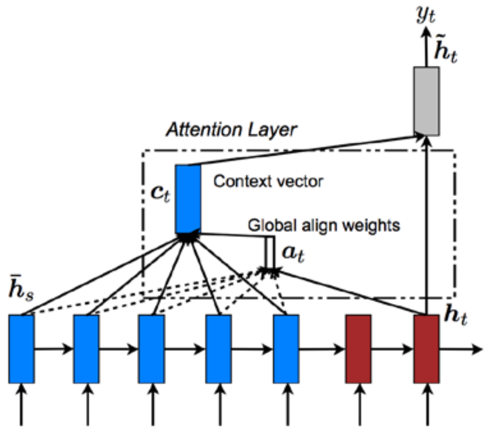
- RNN that iterates through the input cell one token (e.g. code word) at a time, at each time step outputting an “output” vector and a “hidden state” vector.
- The hidden state vector is then passed to the next time step, while the output vector is recorded.
- Multi-layered Gated Recurrent Unit, invented by Cho et al. in 2014.
- Bidirectional- there are essentially two independent RNNs: one that is fed the input sequence in normal sequential order, and one that is fed the input sequence in reverse order. The outputs of each network are summed at each time step. Gives us the advantage of encoding both past and future context.



Bidirectional RNN.
source: https://colah.github.io/posts/2015-09-NN-Types-FP/

Decoder

- RNN that generates the response code lines in a token-by-token fashion. It uses the encoder’s context vectors, and internal hidden states to generate the next word in the sequence. It continues generating words until it outputs an EOS_token (end-of-sentence)
- ATTN - A common problem with a vanilla seq2seq decoder is that if we rely solely on the context vector to encode the entire input sequence’s meaning, we lose information. Especially with long input sequences. To prevent this we use an “global attention mechanism” that allows the decoder to pay attention to certain parts of the input sequence, rather than using the entire fixed context at every step (attention weights). We consider all of the encoder’s hidden states.

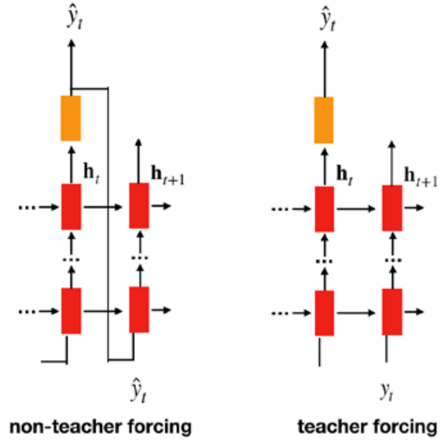


Global attention mechanism.
source: https://arxiv.org/pdf/1508.04025.pdf

OUR **MODEL**

Teacher Forcing

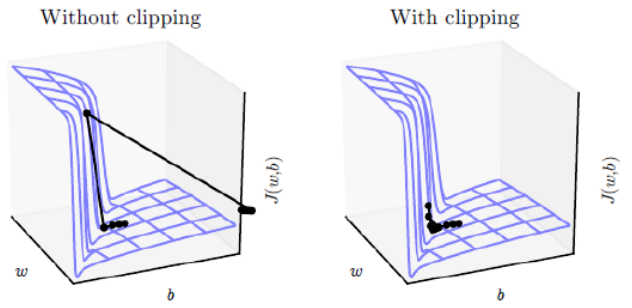
- At some defined probability, we use the current target word as the decoder’s next input rather than using the decoder’s current guess.
- Acts as training wheels for the decoder, aiding in more efficient training.
- May lead to model instability during inference, as the decoder may not have a sufficient chance to truly craft its own output sequences during training. We must be mindful.



source: http://cnyah.com/2017/11/01/professor-forcing/

Gradient clipping

- Used to counter the “exploding gradient” problem.
- By clipping or thresholding gradients to a maximum value, we prevent the gradients from growing exponentially and either overflow (NaN), or overshoot steep cliffs in the cost function.



Gradient clipping example.
source: https://www.deeplearningbook.org/

OUR **MODEL**

Loss

- **Masked Loss**: Since we are dealing with batches of padded sequences, we cannot simply consider all elements of the tensor when calculating loss. We calculate our loss based on our decoder's output tensor, the target tensor, and a binary mask tensor describing the padding of the target tensor.
- **Cross-Entropy**: $H(y,p) = -\sum_i y_i log(p_i)$ - The cross-entropy compares the model's prediction with the real output ("label"). The cross-entropy goes down as the prediction gets more and more accurate, thus it's a good loss function for our needs.
- More on Cross-Entropy

Evaluation

- Along with the loss, we used BLEU score to compare our different next-cell models.
- **BLUE Score**: the **B**i**l**ingual **E**valuation **U**nderstudy, is a score for comparing a candidate text to one or more reference texts. Although developed for translation, it's widely used to evaluate text generated for a suite of NLP tasks.
- More on BLEU

PRE-**PROCESSING**

- Data cleaning:

  As in previous step, “useless” cells has been removed (empty cells/commented cells)

- Created a nicely formatted data file in which each line contains a tab-separated query CELL and a response (Next) CELL pair:

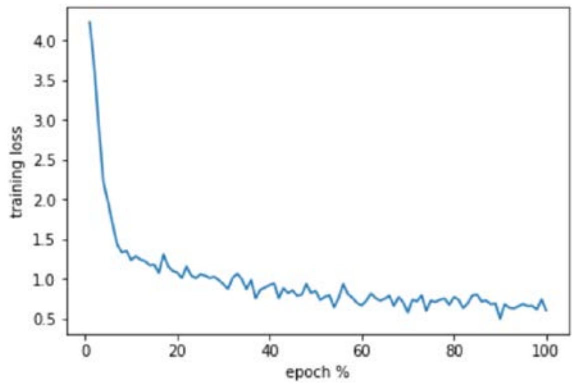
CELL /t *NEXTCELL* /n

```
In [17]: src_pairs = pd.read_csv(src_pairs, sep='\t')
         src_pairs.head(3)
```

Out[17]:

	Source	Next source
0	model.fit(X, y)	test_pred = model.predict(test)
1	test_pred = model.predict(test)	subm['Predicted'] = test_pred subm.to_csv('sub...
2	subm['Predicted'] = test_pred subm.to_csv('sub...	NB_END

FIRST TRY



- Use Pairs of cells source code as input
- Normalized code, separated to tokens, all information kept.
- Fast convergence

Results:

- Low Loss of 0.73
- BLEU score of **0 (!)**
- Not even close…

- Example:

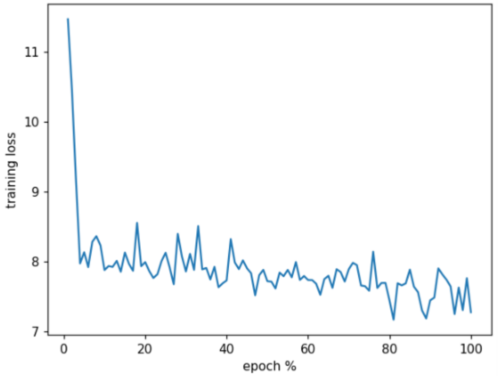
input: df_train = pd.read_csv('../input/20-newsgroups-ciphertext-challenge/train.csv') df_train.head()

output: loadintraining loadintraining imagens imagens fillinh fillinh fillinh faces finance imagens imagens fillinh ysize quarters quarters leftjoin .bandgap datasetexporter krr krr allimage allimage allimage floatcols encipher rod earlystop rod .lotsizesquarefeet bincolumns bincolumns airport startweights startweights hiddenunit atomic imagens imagens imagens .cvr fillinh fillinh faces finance officer cd loadintraining loadintraining titlecat development filtereddf imagens imagens imagens .cvr mhm applicaton loadintraining imagens imagens titlecat imagens .cvr applicaton titlecat filtereddf imagens imagens imagens .cvr applicaton titlecat filtereddf imagens imagens imagens .cvr applicaton titlecat filtereddf imagens imagens imagens .cvr applicaton titlecat filtereddf imagens imagens imagens .cvr applicaton titlecat filtereddf imagens imagens imagens .cvr applicaton titlecat

real next: df_test = pd.read_csv('../input/20-newsgroups-ciphertext-challenge/test.csv') df_test.head()

- Problems:
  - Teacher forcing was 100%, the ”training wheels” didn’t allow the model to learn ⟹ reduce
  - Too much data, Unnecessary ⟹ Remove rare tokens, variable names etc.

SECOND TRY



- Use Pairs of cells source code as input
- Don’t consider rare tokens (still in the data)
- Lower teacher forcing ratio

Results:

- High loss of 7.27 (but now it actually trains)
- BLEU score of 5.76e-232 (better than 0)
- More like it… still pretty bad

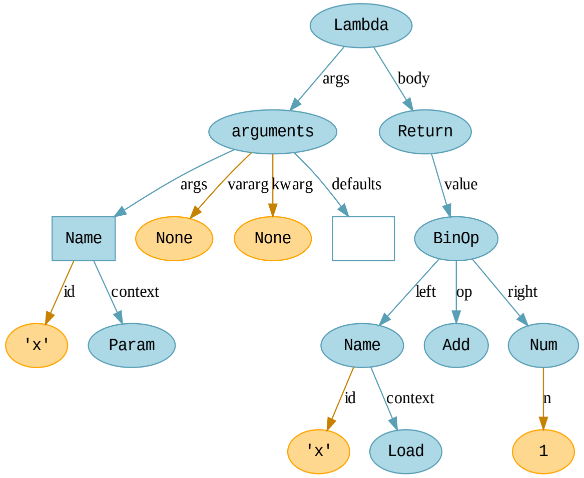
- Example (chatting with the model):

```
> import numpy as np
Bot: train pd .read csv . . input train .tsv sep t test pd .read csv . . input test .tsv sep t np .nan test id
> read_csv(data.csv)
Bot: import matplotlib .pyplot as plt plt .style .use seaborn plt .ylabel frequency . . . . . . . .
> df.head()
Bot: plotpercolumndistribution df np .sum np .sum np .sum np .sum np .mean np .array df .columns df .columns df .columns df
.columns df .columns
```

- Problems:
  - Model gets fixated ⟹ handle repeating tokens, different normalization and representation
  - Out of structure recommendations ⟹ different representation to constraint structure

CODE **AST**

- Abstract Syntax Tree, is a tree representation of the abstract syntactic structure of source code written in a programming language.
- We can understand the structure of a code snippet from its AST
- We can also understand which part is more relevant
- Python ast module
- Example:

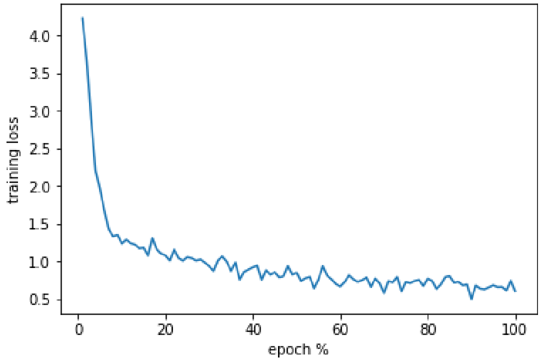



```
df_train = pd.read_csv('../input/20-newsgroups-ciphertext-challenge/train.csv') df_train.head()
```

parse

```
Module(body=[Assign(targets=[Name(id='df_train', ctx=Store())],
                    value=Call(func=Attribute(value=Name(id='pd', ctx=Load()), attr='read_csv', ctx=Load()),
                    args=[Str(s='../input/20-newsgroups-ciphertext-challenge/train.csv')], keywords=[])),
             Expr(value=Call(func=Attribute(value=Name(id='df_train', ctx=Load()), attr='head', ctx=Load()), args=[], keywords=[]))])
```

THIRD TRY



- Use Pairs of cells' AST as input

Results:
- Low loss of 0.64
- BLEU score of 1.17e-231

- Example:

input: Module(body=[Assign(targets=[Name(id='news_list', ctx=Store())],
value=Call(func=Attribute(value=Name(id='pd', ctx=Load()), attr='read_csv', ctx=Load()), args=[Str(s='../input/20-newsgroups/list.csv')], keywords=[])),
Expr(value=Attribute(value=Name(id='news_list', ctx=Load()), attr='shape', ctx=Load()))])

output: module body expr value call func attribute value name id learner ctx load attr fit ctx load args name id lr ctx load num n keywords EOS EOS expr value call func attribute value name id learner ctx load attr fit ctx load args name id lr ctx load num n keywords EOS EOS expr value call func attribute value name id learner ctx load attr fit ctx load args name id lr ctx load num n keywords EOS EOS EOS EOS EOS EOS EOS EOS EOS EOS EOS EOS EOS expr value call func attribute value name id learner

real next: Module(body=[Expr(value=Call(func=Name(id='print', ctx=Load()), args=[Attribute(value=Name(id='df_train', ctx=Load()), attr='shape', ctx=Load()),
Attribute(value=Name(id='df_test', ctx=Load()), attr='shape', ctx=Load())], keywords=[]))])

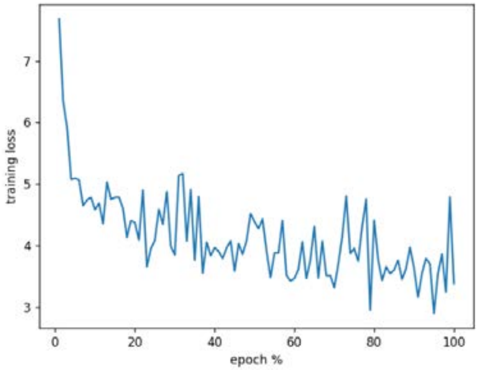
- Problems:
  - Learns structure tokens instead of the recommendations ⟹ different representation

CODE **MASKING**

- We want to use the code's AST to keep only the relevant data for the model
- Tradeoff:
  - Not enough data – too many identical cells and different outputs for same input, can't "reconstruct" the code line to give a working recommendation (for example keeping only function names without any parameters or context)
  - Too much data – model learns irrelevant stuff, bad for prediction (like using the entire AST, the model learns the structure but misses the recommendations)
- We want a summarized representation of just the relevant data

- In addition, we want to keep track of variables, so we can tailor our recommendation to the user (specificalization: output a ready-to-execute recommendation), so when converting the code into the summarized representation we keep a variable dictionary

FOURTH TRY



- Using Masked summed representation of cells

Results:

- Loss of 3.41
- BLEU score of 5.45e-3
- Not too bad, actually some correct recommendations

- Examples:

input: import_datetime import_numpy import_os import_pandas
output: var0=pandas.read_csv
real next: var0=pandas.read_csv var0.head

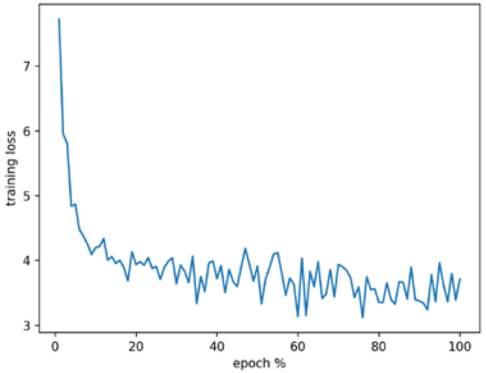
input: var0=pandas.read_csv var0.head
output: var1=pandas.read_csv var1.head
real next: var1=pandas.read_csv var1.head

input: var1=pandas.read_csv var1.head
output: var2=pandas.series matplotlib.pyplot.figure matplotlib.pyplot.hist matplotlib.pyplot.yscale matplotlib.pyplot.title matplotlib.pyplot.xlabel matplotlib.pyplot.ylabel matplotlib.pyplot.ylabel matplotlib.pyplot.ylabel matplotlib.pyplot.ylabel
real next: var2=pandas.read_csv

input: var5.fit
output: nb_end
real next: var6=var3.transform

- Problems:
  - Model eager to end the notebook ⟹ remove empty output cell pairs
  - Long recommendations, doesn't learn well enough where to stop ⟹ line recommendation
  - Got the "easy" ones (the most frequent), what about the rest?

FIFTH TRY



- Input: 3-lines of code, output: next-line-recommendation (instead of input-cell → output-cell)
- Using the same masked representation

Results:

- Loss of 3.71
- BLEU score of 4.03e-232 (less relevant, no n-grams)
- Looks better…

- Examples:

  input: var0=pandas.read_csv var0.head var1=pandas.read_csv
  output: var1.head
  real next: var1.head

  input: var1.head var2=pandas.read_csv matplotlib.pyplot.figure
  output: seaborn.barplot
  real next: seaborn.barplot

  input: var23=lightgbm.Dataset var24=lightgbm.Dataset var25=lightgbm.train
  output: matplotlib.pyplot.figure
  real next: var26=var25.predict

- Problems:
  - Got the “easy” ones (the most frequent), what about the rest? ⟹ use context, split model

RECCOMENDATION **ENGINE**

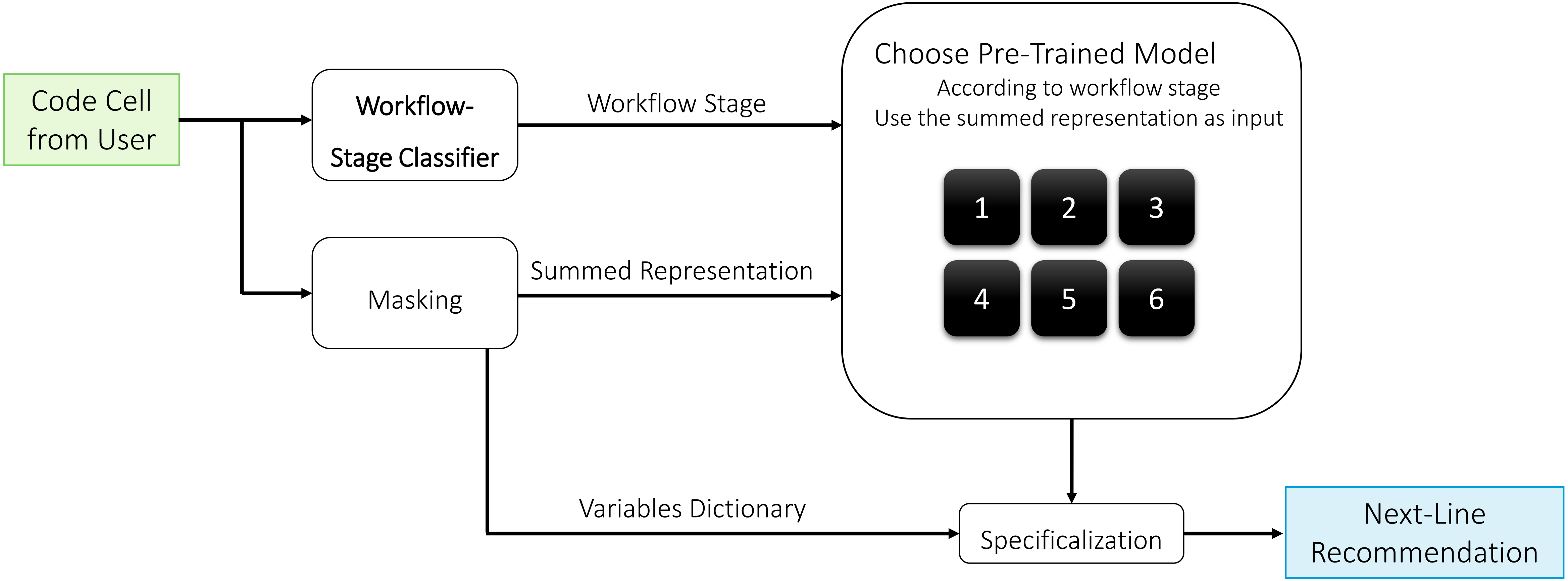
OFFLINE:

- Cells were classified using the workflow stage classifier
- Different model was trained for each stage (of the input cell)

ONLINE:

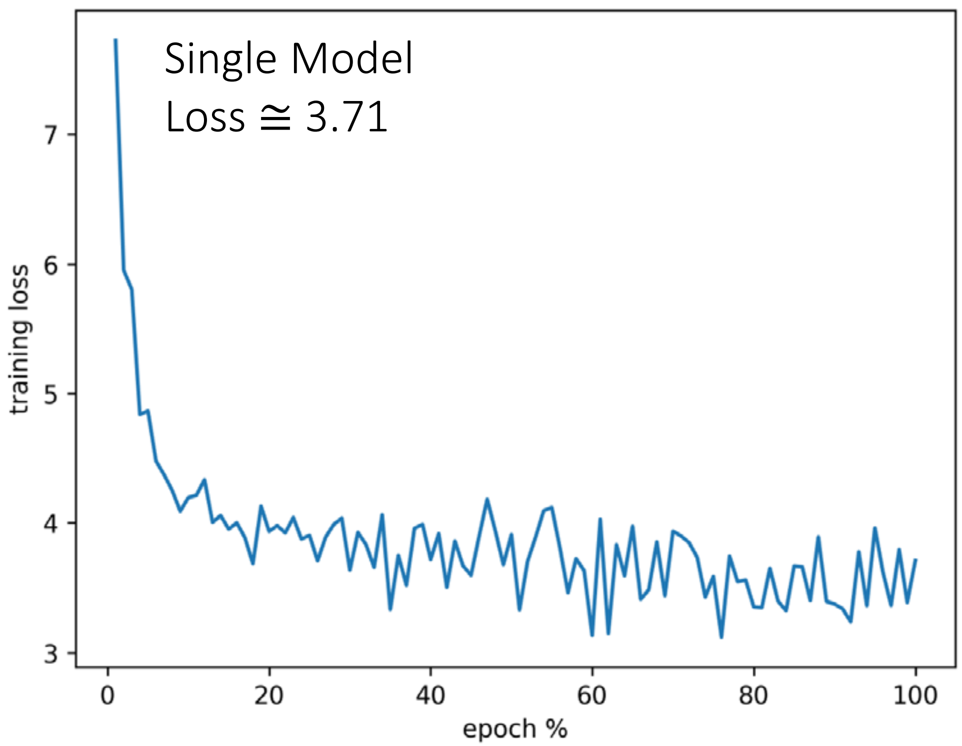
- The user's code is turned into a summed representation using its AST, Variable names are kept in a dictionary
- The user's code is classified and its representation is passed to the relevant model
- The model outputs a next-cell recommendation
- Specificalization: The recommendation is personalized using the Variables dictionary
- Output: a ready-to-execute next cell recommendation

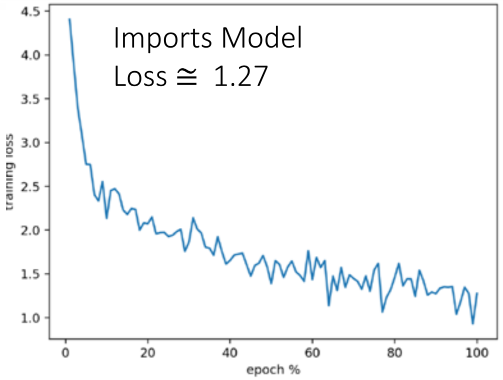
RECCOMENDATION **ENGINE**

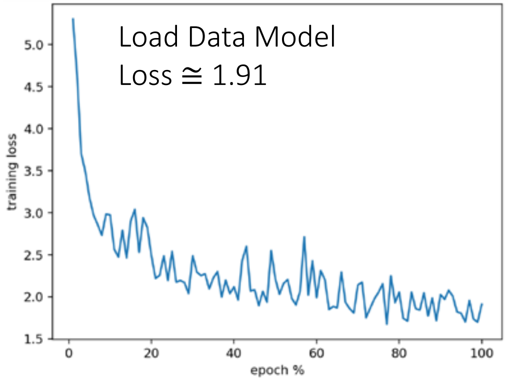


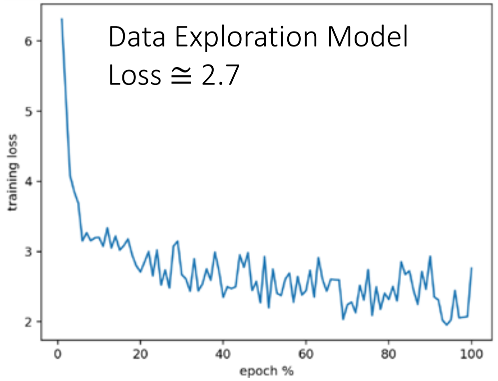
MODEL **EVALUATION**

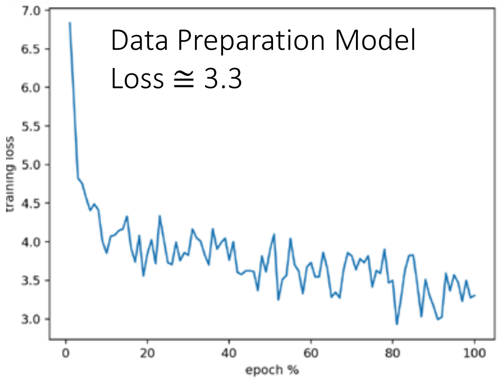
When training a single model for all stages 3-lines → next-line, without context, loss converged at ~3.7, when we used different models according to context we were able to reduce the loss.

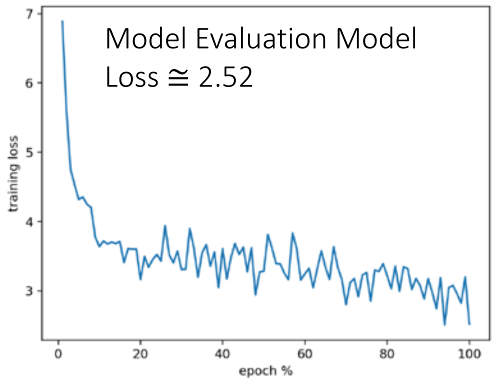












RECOMMENDATIONS **EXAMPLES** (**DEMO**)

FUTURE **WORK**

- Add more data to summarized representation to get more useful recommendations
- Handle loops in next-line recommendation
- Models for Input-stage → Output-stage
- Give recommendation options
- Expand to next cell recommendation
    - Add manual cell structure constraints
    - Handle repeating tokens
- Use more features (outputs?)
- Collect more (and better) data
    - Different sources
    - Grade notebooks, take only good ones
    - Identify forks of same notebook
- Explore different models
- UI integration

THANK YOU