Text Extensions for Pandas

IBM Data Science Community Virtual Webinar
February 24, 2022

Fred Reiss
Principal Research Staff Member, IBM Research
Chief Architect, IBM Center for Open-Source Data and AI Technologies (codait.org)

Hi!

Fred Reiss

Ph.D. from U.C. Berkeley (2006)

Principal Research Staff Member, IBM Research (2006-present)

Chief Architect, IBM Center for Open-Source Data and AI Technologies (2015-present)

In NLP, the easy things are hard.

Example:
Finding facts about people

Joe is the **author** of this document.

This document has positive **sentiment** towards Amy.

This document says that Lois **traveled** to California.

1. Find people
2. Find facts
3. Match facts with people

Example:
Finding facts about people

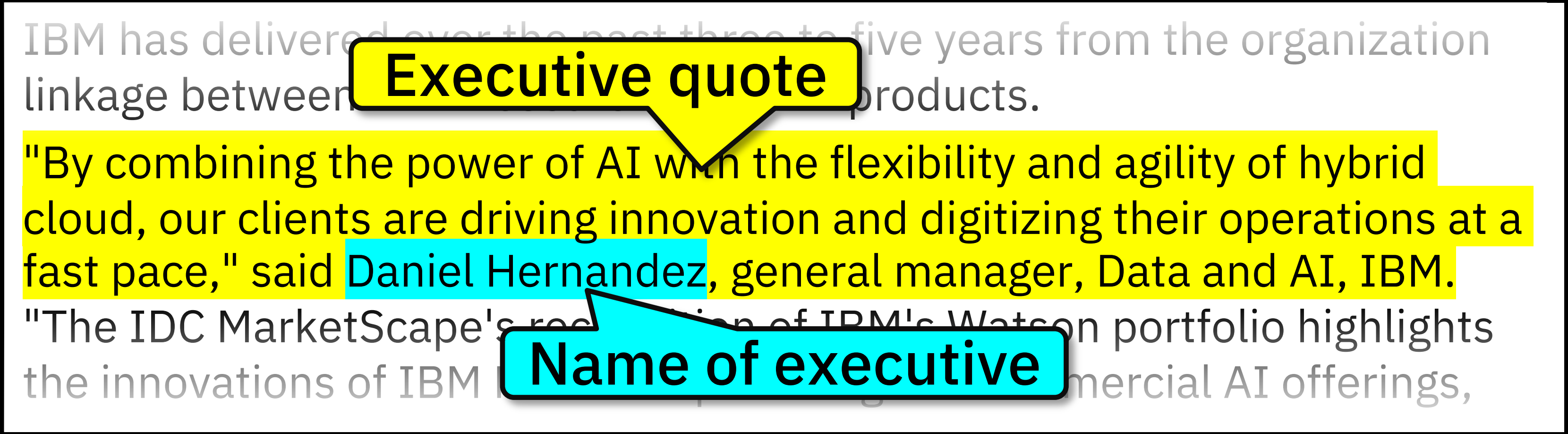
1. Find people
2. Find facts
3. Match facts with people

Hard (Named entity recognition, anaphora resolution, ...)

Hard (Structure identification, sentiment analysis, semantic role labeling...)

Easy

Concrete Example:
Finding Executive Names in Press Releases



Example:
Finding Executive Names in Press Releases

1. Find people
2. Find facts
3. Match facts with people

IBM has delivered over the past three to five years from the organization linkage between IBM Research and IBM products.

"By combining the power of AI with the flexibility and agility of hybrid cloud, our clients are driving innovation and digitizing their operations at a fast pace," said Daniel Hernandez, general manager, Data and AI, IBM. "The IDC MarketScape's recognition of IBM's Watson portfolio highlights the innovations of IBM Research powering our commercial AI offerings,

Example:
Finding Executive Names in Press Releases

1. Find people → Named entity recognition model
2. Find facts (words)
3. Match facts with people

Hugging Face

Load model
Run model
Get results
Filter results

```
ner = transformers.pipeline("ner")
tagged_tokens = ner(document_text)
model_results = ner.group_entities(tagged_tokens)
person_mentions = [d for d in model_results
                   if d["entity_group"] == "PER"]
```

Example:
Finding Executive Names in Press Releases

1. Find people ✅
2. Find facts (quotes)
3. Match facts with people

→ Semantic role labeling model

IBM Watson Natural Language Understanding

Run model

Filter results

Add locations

```
semantic_roles_results = (
    natural_language_understanding
        .analyze(url=doc_url, features=nlu.Features(
            semantic_roles=nlu.SemanticRolesOptions()))
    .get_result()["semantic_roles"])
someone_said_something = [r for r in semantic_roles_results
                          if r["action"]["normalized"] == "say"]
for s in someone_said_something:
    s["subject"]["begin"] = doc_text.find(s["subject"]["text"])
    s["subject"]["end"] = s["subject"]["begin"] + len(s["subject"]["text"])
```

Example:
Finding Executive Names in Press Releases

1. Find people ✅
2. Find facts (quotes) ✅
3. Match facts with people

Compute a chunk size

Build a lookup table

Filter with lookup table and compare pairs of spans

Remove duplicates

Construct a result record

```python
def persons_in_subjects(person_mentions, someone_said_something):
    # Adjust chunk length so every span fits in exactly 1 or 2 chunks
    lengths = [s["subject"]["end"] - s["subject"]["begin"] for s in someone_said_something]
    chunk_len = max(lengths) + 1

    # Build a lookup table.
    # Key is (offset // chunk_len).
    # Value is index into someone_said_something
    chunk_to_srl_ix = {}
    for i in range(len(someone_said_something)):
        s = someone_said_something[i]
        chunk_indices = set(
            [s["subject"]["begin"] // chunk_len, s["subject"]["end"] // chunk_len]
        )
        for chunk_ix in chunk_indices:
            entry = chunk_to_srl_ix.get(chunk_ix, [])
            entry.append(i)
            chunk_to_srl_ix[chunk_ix] = entry

    # Probe into the lookup table and compare pairs of spans
    ix_pairs = []
    for i in range(len(person_mentions)):
        p = person_mentions[i]
        chunk_indices = set([p["start"] // chunk_len, p["end"] // chunk_len])
        ix_to_compare = []
        for chunk_ix in chunk_indices:
            for srl_ix in chunk_to_srl_ix[chunk_ix]:
                srl = someone_said_something[srl_ix]
                if srl["subject"]["begin"] <= p["start"] and srl["subject"]["end"] >= p["end"]:
                    ix_pairs.append((i, srl_ix))

    # Drop duplicates
    unique_ix_pairs = set(ix_pairs)

    # Construct result records
    return [
        {"person": person_mentions[t[0]],
         "subject": someone_said_something[t[1]]["subject"]}
        for t in unique_ix_pairs
    ]
```

In NLP, the **easy things** are hard.

In NLP, the **easy things** are hard.

Building applications with multiple models.

Computing model accuracy.

Creating training data.

Debugging models.

Translating between tokenizations.

...and more

analyzing data

Building applications with multiple models.

Computing model accuracy.

Creating training data.

Debugging models.

Translating between tokenizations.

...and more

Pandas makes **analyzing data** easy.

Building applications with multiple models.

Computing model accuracy.

Creating training data.

Debugging models.

Translating between tokenizations.

…and more

Why use Pandas on NLP data?

1. Transparency
2. Simplicity
3. Compatibility

Demo

If you already use Pandas for NLP…

…you’re probably thinking, “Wait a minute! You can’t do that with Pandas!”

Text Extensions for Pandas

Open-source library from IBM.

Extends Pandas DataFrames to cover natural language processing.

- New data types
- New operations
- Library integrations

Install with pip:

```
pip install text-extensions-for-pandas
```

Install with conda:

```
conda install -c conda-forge \
     text_extensions_for_pandas
```

Source code:
https://github.com/CODAIT/text-extensions-for-pandas

Home page:
https://ibm.biz/text-extensions-for-pandas

Pandas Extension Types

Pandas has a mechanism for adding new types.

Primary extension point: `ExtensionArray`*

Some Pandas built-in types use `ExtensionArray`:

- Timedelta
- Categorical
- Period
- Interval

Subclasses of `ExtensionArray` get:

- Efficient in-memory storage
- Vectorized high-level operations
- Fast binary I/O with Apache Arrow

Text Extensions for Pandas adds two extension types: Spans and Tensors

* More info at `https://pandas.pydata.org/docs/development/extending.html`

Span Data Type (SpanDtype)

```
person_mentions_watson
```

[6]:

	person	confidence
25	[1213, 1229): 'Christoph Herman'	0.992954
30	[2227, 2242): 'Stephen Leonard'	[illegible]

Pandas Series of type SpanDtype

Holds a collection of **spans** (locations in a document).

Internal storage:

- SpanArray object
- Two NumPy arrays (begin and end offsets)
- Shared pointer to document text

Span Data Type (SpanDtype)

```
person_mentions_watson = (
    entity_mentions[entity_mentions["type"] == "Person"]
    [["span", "confidence"]].rename(columns={"span": "person"}))
person_mentions_watson
```

[6]:

	person	confidence
25	[1213, 1229): 'Christoph Herman'	0.992954
30	[2227, 2242): 'Stephen Leonard'	0.995416

Tensor Data Type (`TensorDtype`)

Tensors (n-dimensional arrays) are a crucial part of modern NLP.

`TensorDtype` stores a tensor in each row of a Pandas series

Entire series represented internally with a **single** NumPy array

Many uses: Embeddings, n-grams, image data, time series, ...

embedding
[0.23405041, -0.5534875, 0.9083985, ...
[0.27792975, -0.6853796, 1.1050363, ...
[0.19718906, -0.46341094, 0.5182328, ...
[0.20423545, -0.63758826, 0.82874423, ...
[0.28740737, -0.47174248, 0.77719426, ...

Tensor Data Type (TensorDtype)

	span	ent_iob	ent_type	embedding
70	[155, 168): 'international'	O	<NA>	[0.23405041, -0.5534875, 0.9083985, ...
71	[169, 176): 'between'	O	<NA>	[0.27792975, -0.6853796, 1.1050363, ...
72	[177, 185): 'Pakistan'	B	LOC	[0.19718906, -0.46341094, 0.5182328, ...
73	[186, 189): 'and'	O	<NA>	[0.20423545, -0.63758826, 0.82874423, ...
74	[190, 193): 'New'	B	LOC	[0.28740737, -0.47174248, 0.77719426, ...

Spans of individual tokens

BERT embedding at each token position

Domain-Specific Operations over our Extension Types

```
tp.spanner.contain_join(someone_said_something["subject"],
                        person_mentions_watson["person"],
                        "subject", "person")
```

	subject	person
0	[1213, 1281): 'Christoph Herman, SVP and Head ...	[1213, 1229): 'Christoph Herman'
1	[2227, 2519): 'Stephen Leonard, General Manage...	[2227, 2242): 'Stephen Leonard'

Span-Specific Operations:
Text-specific joins, IOB tagging, gazetteers, …

```
df["X_dot_Y"] = \
  np.sum(df["X"] * df["Y"]).to_numpy()
df
```

	X	Y	X_dot_Y
0	[0.1, 0.2]	[0.5, 0.6]	0.26
1	[0.3, 0.4]	[0.7, 0.3]	0.24

Tensor-Specific Operations:
Most NumPy universal functions

Text Extensions for Pandas: **Input and Output**

Read and write NLP data

- Feather
- Parquet
- Arrow Flight
- PySpark/Dask/Ray serialization
- Universal Dependencies CoNLL-U format

Convert library outputs to DataFrames

- SpaCy
- transformers (from Hugging Face)
- IBM Watson Discovery Table Understanding
- IBM Watson Natural Language Understanding

Example Application: Market Intelligence with Pandas and IBM Watson

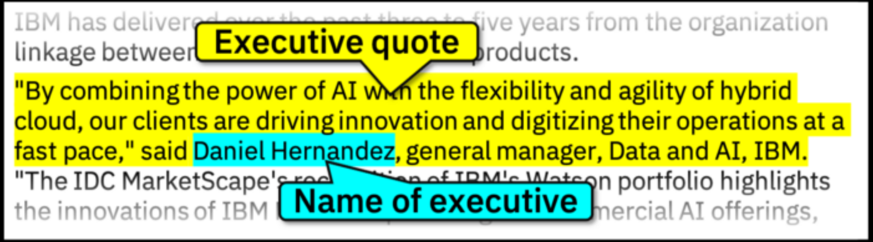
Identify company executives by analyzing press releases.

Tools used:

- Text Extensions for Pandas
- IBM Watson Natural Language Processing

Blog: https://ibm.biz/pandas-market
(Full URL: https://medium.com/ibm-data-ai/market-intelligence-with-pandas-and-ibm-watson-a939323a31ea)

Notebook:
https://ibm.biz/market-notebook
(Full URL: https://github.com/CODAIT/text-extensions-for-pandas/blob/master/tutorials/market/Market_Intelligence_Part1.ipynb)



Identify 301 executives from 191 IBM press releases

	person	url
0	[1977, 1991): 'Wendi Whitmore'	https://newsroom.ibm.com/2020-02-11-IBM-X-Forc...
0	[1281, 1292): 'Rob DiCicco'	https://newsroom.ibm.com/2020-02-18-IBM-Study-...
0	[1213, 1229): 'Christoph Herman'	https://newsroom.ibm.com/2020-02-19-IBM-Power-...
1	[2227, 2242): 'Stephen Leonard'	https://newsroom.ibm.com/2020-02-19-IBM-Power-...
0	[2289, 2297): 'Bob Lord'	https://newsroom.ibm.com/2020-02-26-2020-Call-...
...	...	...
0	[3114, 3124): 'Mike Doran'	https://newsroom.ibm.com/2021-01-25-OVHcloud-t...
0	[3155, 3169): 'Howard Boville'	https://newsroom.ibm.com/2021-01-26-Luminor-Ba...
0	[3114, 3126): 'Samuel Brack'	https://newsroom.ibm.com/2021-01-26-DIA-Levera...
1	[3509, 3523): 'Hillery Hunter'	https://newsroom.ibm.com/2021-01-26-DIA-Levera...
0	[1487, 1497): 'Ana Zamper'	https://newsroom.ibm.com/2021-01-26-Latin-Amer...

301 rows × 2 columns

Text Extensions for Pandas

Open-source library from IBM.

Extends Pandas DataFrames to cover NLP.

- New data types
- New operations
- Library integrations

Install with pip:

```
pip install text-extensions-for-pandas
```

Install with conda:

```
conda install -c conda-forge \
      text_extensions_for_pandas
```

Source code:
https://github.com/CODAIT/text-extensions-for-pandas

https://ibm.biz/text-extensions-for-pandas

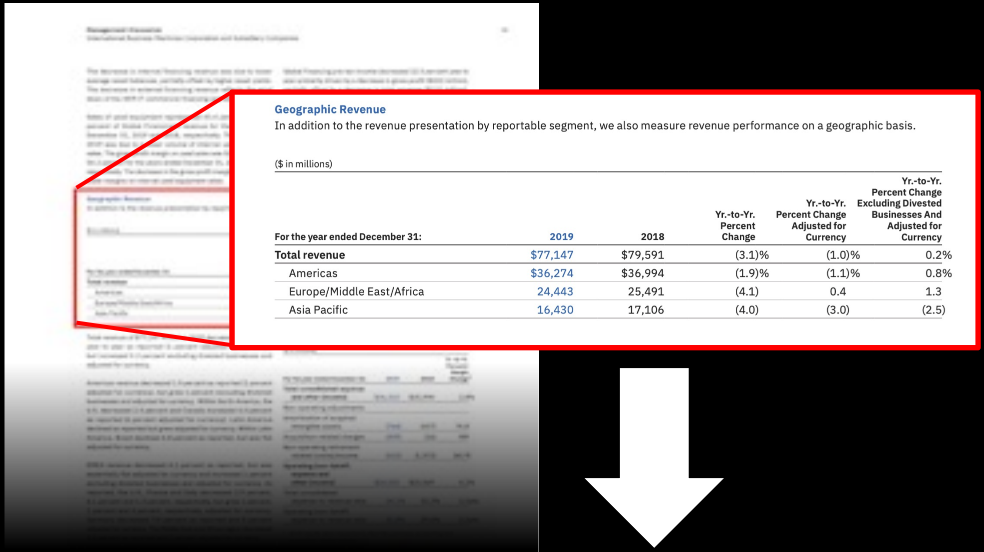
Backup

Example Application: Table Understanding with Pandas and IBM Watson

Extract revenue broken down by geography from 10 years of IBM annual reports in PDF format.

Tools used:
- Text Extensions for Pandas
- IBM Watson Discovery

Notebook:
https://ibm.biz/pandas-table
(https://github.com/CODAIT/text-extensions-for-pandas/blob/master/notebooks/Understand_Tables.ipynb)



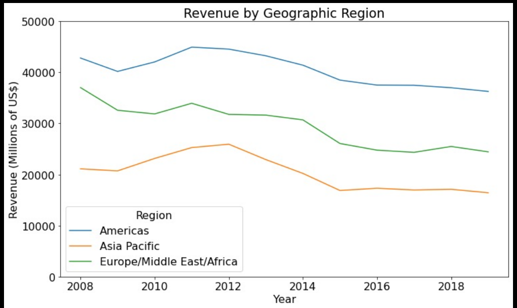
Geographic Revenue

In addition to the revenue presentation by reportable segment, we also measure revenue performance on a geographic basis.

($ in millions)

For the year ended December 31:	2019	2018	Yr.-to-Yr. Percent Change	Yr.-to-Yr. Percent Change Adjusted for Currency	Yr.-to-Yr. Percent Change Excluding Divested Businesses And Adjusted for Currency
Total revenue	$77,147	$79,591	(3.1)%	(1.0)%	0.2%
Americas	$36,274	$36,994	(1.9)%	(1.1)%	0.8%
Europe/Middle East/Africa	24,443	25,491	(4.1)	0.4	1.3
Asia Pacific	16,430	17,106	(4.0)	(3.0)	(2.5)

Year	2008	2009	2010	2011	2012	2013	2014	2015	2016	2017	2018	2019
Region												
Americas	42807.0	40184.0	42044.0	44944.0	44556.0	43249.0	41410.0	38486.0	37513.0	37479.0	36994.0	36274.0
Asia Pacific	21111.0	20710.0	23150.0	25273.0	25937.0	22923.0	20216.0	16871.0	17313.0	16970.0	17106.0	16430.0
Europe/Middle East/Africa	37020.0	32583.0	31866.0	33952.0	31775.0	31628.0	30700.0	26073.0	24769.0	24345.0	25491.0	24443.0



transformers

Retokenize with the BERT tokenizer and translate inside-outside-beginning (IOB) tags from the corpus's original tokenization.

```
[19]: # Generate IOB2 tags and entity labels that align with the BERT tokens.
      # See https://en.wikipedia.org/wiki/Inside%E2%80%93outside%E2%80%93beginning_(tagging)
      bert_toks_df[["ent_iob", "ent_type"]] = tp.io.conll.spans_to_iob(bert_spans,
                                                                        spans_df["ent_type"])
      bert_toks_df[10:20]
```

[19]:

	token_id	span	input_id	token_type_id	attention_mask	special_tokens_mask	ent_iob	ent_type
10	10	[15, 17): 'KE'	22441	0	1	False	O	<NA>
11	11	[17, 18): 'T'	1942	0	1	False	O	<NA>
12	12	[18, 19): '-'	118	0	1	False	O	<NA>
13	13	[20, 22): 'PA'	8544	0	1	False	B	LOC
14	14	[22, 23): 'K'	2428	0	1	False	I	LOC
15	15	[23, 25): 'IS'	6258	0	1	False	I	LOC
16	16	[25, 27): 'TA'	9159	0	1	False	I	LOC
17	17	[27, 28): 'N'	2249	0	1	False	I	LOC
18	18	[29, 30): 'V'	159	0	1	False	O	<NA>
19	19	[31, 33): 'NE'	26546	0	1	False	B	LOC

transformers

Compute BERT embeddings and add a column of tensors containing the embedding for the window around each token position.

```
[24]: # Initialize the BERT model that will be used to generate embeddings.
bert = transformers.BertModel.from_pretrained(bert_model_name)

# Force garbage collection in case this notebook is running on a low-RAM environment.
gc.collect()

# Compute BERT embeddings with the BERT model and add result to our example DataFrame.
embeddings_df = tp.io.bert.add_embeddings(classes_df, bert)
embeddings_df[["token_id", "span", "input_id", "ent_iob", "ent_type", "token_class", "embedd
```

	token_id	span	input_id	ent_iob	ent_type	token_class	embedding
10	10	[15, 17): 'KE'	22441	O	<NA>	O	[-0.19854125, -0.46898478, 0.7755599...
11	11	[17, 18): 'T'	1942	O	<NA>	O	[-0.24190304, -0.42399377, 0.955406...
12	12	[18, 19): '-'	118	O	<NA>	O	[-0.20076738, -0.7481939, 1.302213...
13	13	[20, 22): 'PA'	8544	B	LOC	B-LOC	[0.2020257, -0.26199907, 0.3297634...
14	14	[22, 23): 'K'	2428	I	LOC	I-LOC	[-0.5462166, -0.90924495, -0.05836733...
15	15	[23, 25): 'IS'	6258	I	LOC	I-LOC	[-0.37400314, -0.6890743, -0.1446248...
16	16	[25, 27): 'TA'	9159	I	LOC	I-LOC	[-0.46548596, -0.8717423, 0.3557480...
17	17	[27, 28): 'N'	2249	I	LOC	I-LOC	[-0.18682732, -0.9008188, 0.3601504...
18	18	[29, 30): 'V'	159	O	<NA>	O	[-0.16640136, -0.8363809, 0.874061...
19	19	[31, 33): 'NE'	26546	B	LOC	B-LOC	[-0.3024105, -0.8382667, 1.105809...

SpaCy

Convert the output of a SpaCy language model into a DataFrame.

```
[19]: doc_text = response["analyzed_text"]
      spacy_language_model = spacy.load("en_core_web_sm")
      token_features = tp.io.spacy.make_tokens_and_features(doc_text, spacy_language_model)
      token_features
```

[19]:

	id	span	lemma	pos	tag	dep	head	shape	ent_iob	ent_type	is_alpha	is_stop
0	0	[0, 2): 'In'	in	ADP	IN	prep	12	Xx	O		True	True
1	1	[3, 5): 'AD'	ad	NOUN	NN	pobj	0	XX	B	DATE	True	False
2	2	[6, 9): '932'	932	NUM	CD	nummod	1	ddd	I	DATE	False	False
3	3	[9, 10): ','	,	PUNCT	,	punct	12	,	O		False	False
4	4	[11, 15): 'King'	King	PROPN	NNP	compound	5	Xxxx	B	ORG	True	False

SpaCy

Use Pandas to select part of the document, then display the dependency parse for just that part of the document using DisplayCy.