tiny-blocks

Jose-Maria Vazquez-Jimenez

Sep 09, 2022

CONTENTS:

Tiny Blocks to build large and complex ETL pipelines!

Tiny-Blocks is a library for **data engineering** operations. Each **pipeline** is made out of **tiny-blocks** glued with the >> operator. This library relies on a fundamental streaming abstraction consisting of three parts: **extract**, **transform**, and **load**. You can view a pipeline as an extraction, followed by zero or more transformations, followed by a sink. Visually, this looks like:

```
extract -> transform1 -> transform2 -> ... -> transformN -> load
```

You can also *fan-in*, *fan-out* for more complex operations:

```
extract1 -> transform1 -> |-> transform2 -> ... -> | -> transformN -> load1
extract2 ----------------> |                       | -> load2
```

Tiny-Blocks use **generators** to stream data. Each **chunk** is a **Pandas DataFrame**. The *chunksize* or buffer size is adjustable per pipeline.

CHAPTER

ONE

BASIC USAGE

Make sure you had install the package by doing `pip install tiny-blocks` and then:

```
from tiny_blocks.extract import FromCSV
from tiny_blocks.transform import Fillna
from tiny_blocks.load import ToSQL

# ETL Blocks
from_csv = FromCSV(path='/path/to/source.csv')
fill_na = Fillna(value="Hola Mundo")
to_sql = ToSQL(dsn_conn='psycopg2+postgres://...', table_name="sink")

# Pipeline
from_csv >> fill_na >> to_sql
```

1.1 Installation

Install tiny-blocks:

```
pip install tiny-blocks
```

1.1.1 Advanced: local setup for development (Ubuntu)

These instructions assume that `git`, `docker`, and `docker-compose` are installed on your host machine.

First, clone this repo and make some required directories.:

```
git clone https://github.com/pyprogrammerblog/tiny-blocks.git
cd tiny-blocks
```

Then build the docker image:

```
docker-compose run --rm app poetry install
```

Then install dependencies:

```
docker-compose run --rm app poetry install
```

Run the tests:

```
docker-compose run app poetry run pytest
```

Then run the app and access inside the docker with the env activated:

```
docker-compose run --rm app poetry shell
```

Finally you can down the services:

```
docker-compose down
```

1.1.2 Advanced: Jupyter Notebook

Hit the command:

```
docker-compose run --rm -p 8888:8888 app poetry shell
```

Then inside the docker:

```
jupyter notebook --ip 0.0.0.0 --no-browser --allow-root
```

1.2 Tiny-Blocks

1.2.1 FanIn

1.2.2 FanOut

1.3 Extract Operations

1.3.1 ExtractBase

class tiny_blocks.extract.base.**ExtractBase**(*, *uuid: UUID = None*, *name: str*, *version: str = 'v1'*, *description: str = None*)

Extract Base Block.

Each extraction Block implement the *get_iter* method. This method return an Iterator of chunked DataFrames

get_iter() → Iterator[DataFrame]

Return an iterator of chunked dataframes

The *chunksize* is defined as kwargs in each extraction block

1.3.2 FromCSV

class tiny_blocks.extract.from_csv.**FromCSV**(*, *uuid: UUID = None*, *name: Literal['read_csv'] = 'read_csv'*, *version: str = 'v1'*, *description: str = None*, *path: pydantic.types.FilePath | pydantic.networks.AnyUrl*, *kwargs: KwargsFromCSV = KwargsFromCSV(sep='|', header='infer', names=None, index_col=None, usecols=None, squeeze=False, prefix=None, mangle_dupe_cols=True, dtype=None, converters=None, engine=None, true_values=None, false_values=None, chunksize=1000, storage_options=None, skipinitialspace=False, skiprows=None, skipfooter=None, nrows=None, na_values=None, keep_default_na=True, na_filter=True, verbose=False, skip_blank_lines=True, parse_dates=None, infer_datetime_format=False, keep_date_col=False, date_parser=None, dayfirst=False, cache_dates=True, compression='infer', thousands=None, decimal='.', lineterminator=None, quotechar=None, quoting=None, doublequote=True, escapechar=None, comment=None, encoding=None, encoding_errors='strict', dialect=None, on_bad_lines='skip', delim_whitespace=False, low_memory=True, memory_map=False, float_precision=None)*)

ReadCSV Block. Defines the read CSV Operation

Basic example:

```
>>> import pandas as pd
>>> from tiny_blocks.extract import FromCSV
>>>
>>> read_csv = FromCSV(path="/path/to/file.csv")
>>>
>>> generator = read_csv.get_iter()
>>> df = pd.concat(generator)
```

See info about Kwargs: https://pandas.pydata.org/docs/reference/api/pandas.read_csv.html

1.3.3 FromSQLTable

class tiny_blocks.extract.from_sql_table.**FromSQLTable**(*, *uuid: UUID = None*, *name: Literal['read_sql_table'] = 'read_sql_table'*, *version: str = 'v1'*, *description: str = None*, *dsn_conn: str*, *table_name: str*, *kwargs: KwargsFromSQLTable = KwargsFromSQLTable(schma=None, index_col=None, coerce_float=True, parse_dates=None, columns=None, chunksize=1000)*)

Read SQL Table Block. Defines the read SQL Table Operation.

Basic example:

```
>>> import pandas as pd
>>> from tiny_blocks.extract import FromSQLTable
>>>
>>> str_conn = "postgresql+psycopg2://user:pass@postgres:5432/db"
>>> read_sql = FromSQLTable(dsn_conn=str_conn, table_name="test")
>>>
>>> generator = read_sql.get_iter()
>>> df = pd.concat(generator)
```

See info about Kwargs: https://pandas.pydata.org/docs/reference/api/pandas.read_sql_table.html

connect_db() → Connection

Opens a DB transaction. Yields a connection to Database defined in *dsn_conn*.

Parameters set on the connection are:

- *autocommit* mode set to *True*.
- Connection mode *stream_results* set as *True*.

1.3.4 FromSQLQuery

class tiny_blocks.extract.from_sql_query.**FromSQLQuery**(*, *uuid: UUID = None*, *name: Literal['read_sql'] = 'read_sql'*, *version: str = 'v1'*, *description: str = None*, *dsn_conn: str*, *sql: str*, *kwargs: KwargsFromSQLQuery = KwargsFromSQLQuery(index_col=None, coerce_float=True, params=None, parse_dates=None, chunksize=1000, dtype=None)*)

Read SQL Query Block. Defines the read SQL Query Operation

Basic example:

```
>>> import pandas as pd
>>> from tiny_blocks.extract import FromSQLQuery
>>>
>>> str_conn = "postgresql+psycopg2://user:pass@postgres:5432/db"
>>> sql = "select * from test"
>>> read_sql = FromSQLQuery(dsn_conn=str_conn, sql=sql)
>>>
>>> generator = read_sql.get_iter()
>>> df = pd.concat(generator)
```

See info about Kwargs: https://pandas.pydata.org/docs/reference/api/pandas.read_sql_query.html

connect_db() → Connection

Opens a DB transaction. Yields a connection to Database defined in *dsn_conn*.

Parameters set on the connection are:

- *autocommit* mode set to *True*.
- Connection mode *stream_results* set as *True*.

1.3.5 FromKafka

class tiny_blocks.extract.from_kafka.**FromKafka**(*, *uuid: UUID = None*, *name: Literal['from_kafka'] = 'from_kafka'*, *version: str = 'v1'*, *description: str = None*, *kwargs: KwargsFromKafka = KwargsFromKafka(consumer_timeout=1000)*, *topic: str*, *group_id: str*, *bootstrap_servers: List[str]*)

FromKafka Block. Defines the read Kafka Operation

kafka_consumer() → KafkaConsumer

Yields a consumer to a Kafka topic.

Parameters set on the connection are:

- *topic*.
- *group_id*.
- *bootstrap_servers*. List of server strings.
- *auto_offset_reset* is set to *True*.
- *enable_auto_commit* is set to *True*.
- *consumer_timeout_ms* by default to 1 second.

1.4 Transform Operations

1.4.1 TransformBase

class tiny_blocks.transform.base.**TransformBase**(*, *uuid: UUID = None*, *name: str*, *version: str = 'v1'*, *description: str = None*)

Transform Base Block

Each transformation Block implements the *get_iter* method. This method get one or multiple iterators and return an Iterator of chunked DataFrames.

get_iter(*source*) → Iterator[DataFrame]

Return an iterator of chunked dataframes

The *chunksize* is defined as kwargs in each transformation block

1.4.2 Apply

class tiny_blocks.transform.apply.**Apply**(*, *uuid: UUID = None*, *name: Literal['apply'] = 'apply'*, *version: str = 'v1'*, *description: str = None*, *apply_to_column: str*, *set_to_column: str*, *func: Callable*, *kwargs: KwargsApply = KwargsApply()*)

Apply function. Defines block to apply function.

The method is applied to a single column. For different functionality please rewrite the Block.

Basic example:

```
>>> import pandas as pd
>>> from tiny_blocks.transform import Apply
>>> from tiny_blocks.extract import FromCSV
>>>
>>> from_csv = FromCSV(path='/path/to/file.csv')
>>> apply = Apply(
...   apply_to_column="column_A",
...   set_to_column="column_b",
...   func=lambda x: x + 1,
>>> )
>>>
>>> generator = from_csv.get_iter()
>>> generator = apply.get_iter(generator)
>>> df = pd.concat(generator)
```

For more Kwargs info: https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.apply.html

1.4.3 Astype

class tiny_blocks.transform.astype.**Astype**(*, *uuid: UUID = None*, *name: Literal['astype'] = 'astype'*, *version: str = 'v1'*, *description: str = None*, *dtype: Dict[str, str]*, *kwargs: KwargsAstype = KwargsAstype(errors='ignore')*)

Astype Block. Defines the type casting for column dataframes.

Basic example:

```
>>> import pandas as pd
>>> from tiny_blocks.transform import Astype
>>> from tiny_blocks.extract import FromCSV
>>>
>>> from_csv = FromCSV(path="/path/to/file.csv")
>>> as_type = Astype(dtype={"e": "float32"})
>>>
>>> generator = from_csv.get_iter()
>>> generator = as_type.get_iter(generator)
>>> df = pd.concat(generator)
```

For more Kwargs info: https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.astype.html

1.4.4 DropDuplicates

class tiny_blocks.transform.drop_duplicates.**DropDuplicates**(*, *uuid: UUID = None*, *name: Literal['drop_duplicates'] = 'drop_duplicates'*, *version: str = 'v1'*, *description: str = None*, *kwargs: KwargsDropDuplicates = KwargsDropDuplicates(chunksize=1000)*, *keep: Literal['first', 'last'] = 'first'*, *subset: Set[str] = None*)

Drop Duplicates Block. Defines the drop duplicates functionality

Basic example:

```
>>> import pandas as pd
>>> from tiny_blocks.transform import DropDuplicates
>>> from tiny_blocks.extract import FromCSV
>>>
>>> extract_csv = FromCSV(path='/path/to/file.csv')
>>> drop_duplicates = DropDuplicates()
>>>
>>> generator = extract_csv.get_iter()
>>> generator = drop_duplicates.get_iter(generator)
>>> df = pd.concat(generator)
```

1.4.5 DropNa

class tiny_blocks.transform.dropna.**DropNa**(*, *uuid: UUID = None*, *name: Literal['drop_na'] = 'drop_na'*, *version: str = 'v1'*, *description: str = None*, *kwargs: KwargsDropNa = KwargsDropNa(subset=None, axis=None, how=None, thresh=None)*)

Drop Nan Block. Defines the drop None values functionality

Basic example:

```
>>> import pandas as pd
>>> from tiny_blocks.transform import DropNa
>>> from tiny_blocks.extract import FromCSV
>>>
>>> extract_csv = FromCSV(path='/path/to/file.csv')
>>> drop_na = DropNa()
>>>
>>> generator = extract_csv.get_iter()
>>> generator = drop_na.get_iter(generator)
>>> df = pd.concat(generator)
```

For more Kwargs info: https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.dropna.html

1.4.6 Merge

class tiny_blocks.transform.merge.**Merge**(*, *uuid: UUID = None*, *name: Literal['merge'] = 'merge'*, *version: str = 'v1'*, *description: str = None*, *how: Literal['left', 'right', 'outer', 'inner', 'cross'] = 'inner'*, *left_on: str*, *right_on: str*, *kwargs: KwargsMerge = KwargsMerge(chunksize=1000)*)

Merge. Defines merge functionality between two blocks.

Basic example:

```
>>> import pandas as pd
>>> from tiny_blocks.transform import Merge
>>> from tiny_blocks.extract import FromCSV
>>>
>>> from_csv_1 = FromCSV(path="/path/to/file_1.csv")
>>> from_csv_2 = FromCSV(path="/path/to/file_2.csv")
```

(continues on next page)

(continued from previous page)

```
>>> merge = Merge(how="left", left_on="col_A", right_on="col_B")
>>>
>>> left_source = from_csv_1.get_iter()
>>> right_source = from_csv_2.get_iter()
>>> generator = merge.get_iter(source=[left_source, right_source])
>>> df = pd.concat(generator)
```

1.4.7 Rename

class tiny_blocks.transform.rename.**Rename**(*, *uuid: UUID = None*, *name: Literal['rename'] = 'rename'*, *version: str = 'v1'*, *description: str = None*, *kwargs: KwargsRename = KwargsRename(axis=None, level=None, errors=None)*, *columns: Dict[str, str]*)

Rename Block. Defines Rename columns functionality

Basic example:

```
>>> import pandas as pd
>>> from tiny_blocks.transform import Rename
>>> from tiny_blocks.extract import FromCSV
>>>
>>> from_csv = FromCSV(path='/path/to/file.csv')
>>> sort = Rename(columns={"column_name": "new_column_name"})
>>>
>>> generator = from_csv.get_iter()
>>> generator = sort.get_iter(generator)
>>> df = pd.concat(generator)
```

For more Kwargs info: https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.rename.html

1.4.8 Sort

class tiny_blocks.transform.sort.**Sort**(*, *uuid: UUID = None*, *name: Literal['sort'] = 'sort'*, *version: str = 'v1'*, *description: str = None*, *by: List[str]*, *ascending: bool = True*, *kwargs: KwargsSort = KwargsSort(chunksize=1000)*)

Sort Block. Defines the Sorting operation

Basic example:

```
>>> import pandas as pd
>>> from tiny_blocks.transform import Sort
>>> from tiny_blocks.extract import FromCSV
>>>
>>> extract_csv = FromCSV(path='/path/to/file.csv')
>>> sort = Sort(by=["column_A"], ascending=False)
>>>
>>> generator = extract_csv.get_iter()
>>> generator = sort.get_iter(generator)
>>> df = pd.concat(generator)
```

1.4.9 Validate

class tiny_blocks.transform.validate.**Validate**(*, *uuid: UUID = None*, *name: Literal['validate'] = 'validate'*, *version: str = 'v1'*, *description: str = None*, *schema_model: SchemaModel*, *lazy: bool = True*)

Validate block. Defines block to apply validation.

Basic example:

```
>>> import pandas as pd
>>> from tiny_blocks.transform import Apply
>>> from tiny_blocks.extract import FromCSV
>>>
>>> from_csv = FromCSV(path='/path/to/file.csv')
>>> validate = Validate(
...   schema_model=my_schema_validation, lazy=True
>>> )
>>>
>>> generator = from_csv.get_iter()
>>> generator = validate.get_iter(generator)
>>> df = pd.concat(generator)
```

1.5 Load Operations

1.5.1 LoadBase

class tiny_blocks.load.base.**LoadBase**(*, *uuid: UUID = None*, *name: str*, *version: str = 'v1'*, *description: str = None*)

Load Base Block

All blocks inheriting the LoadBase class must implement the *exhaust* method.

exhaust(*source: Iterator[DataFrame]*)

Implement the exhaustion of the incoming iterator.

It is the end of the Pipe.

1.5.2 ToCSV

class tiny_blocks.load.to_csv.**ToCSV**(*, *uuid: UUID = None*, *name: Literal['to_csv'] = 'to_csv'*, *version: str = 'v1'*, *description: str = None*, *kwargs: KwargsToCSV = KwargsToCSV(sep='|', na_rep=None, float_format=None, columns=None, header=True, index=False, index_label=None, mode=None, encoding=None, compression='infer', quoting=None, quotechar=None, line_terminator=None, chunksize=1000, date_format=None, doublequote=None, escapechar=None, decimal=None, errors=None, storage_options=None)*, *path: pathlib.Path | pydantic.networks.AnyUrl*)

Write CSV Block. Defines the load to CSV Operation

Basic example:

```
>>> from tiny_blocks.load import ToCSV
>>> from tiny_blocks.extract import FromCSV
>>>
>>> from_csv = FromCSV(path="path/to/source.csv")
>>> to_csv = ToCSV(path="path/to/sink.csv")
>>>
>>> generator = from_csv.get_iter()
>>> to_csv.exhaust(generator)
```

See info about Kwargs: https://pandas.pydata.org/docs/reference/api/pandas.to_csv.html

1.5.3 ToSQL

class tiny_blocks.load.to_sql.**ToSQL**(*, *uuid: UUID = None*, *name: Literal['to_sql'] = 'to_sql'*, *version: str = 'v1'*, *description: str = None*, *dsn_conn: str*, *table_name: str*, *kwargs: KwargsToSQL = KwargsToSQL(schma=None, if_exists='append', index=False, index_label=None, dtype=None, chunksize=1000, method=None)*)

Load SQL Block. Defines the Loading operation to a SQL Database

Basic example:

```
>>> from tiny_blocks.extract import FromSQLTable
>>> from tiny_blocks.load import ToSQL
>>>
>>> str_conn = "postgresql+psycopg2://user:pass@postgres:5432/db"
>>> from_sql = FromSQLTable(dsn_conn=str_conn, table_name="source")
>>> to_sql = ToSQL(dsn_conn=str_conn, table_name="sink")
>>>
>>> generator = from_sql.get_iter()
>>> to_sql.exhaust(generator)
```

For more Kwargs info: https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.to_sql.html

connect_db() → Connection

Opens a DB transaction. Yields a connection to Database defined in *dsn_conn*.

Parameters set on the connection are:

- *autocommit* mode set to *True*.
- Connection mode *stream_results* set as *True*.

1.5.4 ToKafka

class tiny_blocks.load.to_kafka.**ToKafka**(*, *uuid: UUID = None*, *name: Literal['to_kafka'] = 'to_kafka'*, *version: str = 'v1'*, *description: str = None*, *kwargs: KwargsToKafka = KwargsToKafka(consumer_timeout=1000)*, *topic: str*, *group_id: str*, *bootstrap_servers: List[str]*)

Write CSV Block. Defines the load to CSV Operation

kafka_producer() → KafkaProducer

Yields a Producer.

Parameters set on the connection are:

- *group_id*.
- *bootstrap_servers*. List of server strings.
- *auto_offset_reset* is set to *True*.
- *enable_auto_commit* is set to *True*.
- *consumer_timeout_ms* by default to 1 second.

1.6 License

1.7 Any help

Everyone is encouraged to file bug reports, feature requests, and pull requests through GitHub.

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

t

INDEX

V