AnkiPandas

Release 0.3.4

Kilian Lieret

Jul 09, 2020

Basics

Load your anki database as a pandas DataFrame with just one line of code!

CHAPTER 1

Readme

NOTE: THIS PROJECT IS STILL RATHER FRESH. TRY OUT WITH CARE AND GIVE FEEDBACK!

1.1 Description

Anki is one of the most popular flashcard system for spaced repetition learning, pandas is the most popular python package for data analysis and manipulation. So what could be better than to bring both together?

With `AnkiPandas` you can use `pandas` to easily analyze or manipulate your Anki flashcards.

Features:

- **Select**: Easily select arbitrary subsets of your cards, notes or reviews using `pandas` (one of many introductions, official documentation)
- **Visualize**: Use pandas' powerful built in tools or switch to the even more versatile seaborn (statistical analysis) or matplotlib libraries
- **Manipulate**: Apply fast bulk operations to the table (e.g. add tags, change decks, set field contents, suspend cards, ...) or iterate over the table and perform these manipulations step by step
- **Add**: Add new notes and cards
- **Import and Export**: Pandas can export to (and import from) csv, MS Excel, HTML, JSON, ... (io documentation)

Pros:

- **Easy installation**: Install via python package manager (independent of your Anki installation)

- **Simple**: Just one line of code to get started
- **Convenient**: Bring together information about cards, notes, models, decks and more in just one table!
- **Fully documented**: Documentation on readthedocs
- **Well tested**: More than 100 unit tests to keep everything in check

Alternatives: If your main goal is to add new cards, models and more, you can also take a look at the genanki project.

1.2 Installation

`AnkiPandas` is available as pypi package and can be installed or upgrade with the python package manager:

```
pip3 install --user --upgrade ankipandas
```

For the latest development version you can also work from a cloned version of this repository:

```
git clone https://github.com/klieret/ankipandas/
cd ankipandas
pip3 install --user --upgrade .
```

1.3 Usage

Starting up is as easy as this:

```
from ankipandas import Collection

col = Collection()
```

And `col.notes` will be dataframe containing all notes, with additional methods that make many things easy. Similarly, you can access cards or reviews using `col.cards` or `col.revs`.

If called without any argument `Collection()` tries to find your Anki database by itself. However this might take some time. To make it easier, simply supply (part of) the path to the database and (if you have more than one user) your Anki user name, e.g. `Collection(".local/share/Anki2/", user="User 1")` on many Linux installations.

To get information about the interpretation of each column, use `print(col.notes.help_cols())`.

Take a look at the documentation to find out more about more about the available methods!

Some basic examples:

1.3.1 Analysis

** More examples can be found in the analysis documentation **

Show a histogram of the number of reviews (repetitions) of each card for all decks:

```
col.cards.hist(column="creps", by="cdeck")
```

Show the number of leeches per deck as pie chart:

```
cards = col.cards.merge_notes()
selection = cards[cards.has_tag("leech")]
selection["cdeck"].value_counts().plot.pie()
```

Find all notes of model `MnemoticModel` with empty `Mnemotic` field:

```
notes = col.notes.fields_as_columns()
notes.query("model=='MnemoticModel' and 'Mnemotic'==''")
```

1.3.2 Manipulations

Add the `difficult-japanese` and `marked` tag to all notes that contain the tags `Japanese` and `leech`:

```
selection = col.notes.has_tags(["Japanese", "leech"])
selection = selection.add_tag(["difficult-japanese", "marked"])
col.notes.update(selection)
col.write(modify=True)  # Overwrites your database after creating a backup!
```

Set the `language` field to `English` for all notes of model `LanguageModel` that are tagged with `English`:

```
selection = col.notes.has_tag(["English"]).query("model=='LanguageModel'").copy()
fields_as_columns(inplace=True)
selection["language"] = "English"
col.notes.update(selection)
col.write(modify=True)
```

Move all cards tagged `leech` to the deck `Leeches Only`:

```
selection = col.cards.has_tag("leech")
selection["cdeck"] = "Leeches Only"
col.cards.update(selection)
col.write(modify=True)
```

1.4 Troubleshooting

See the troubleshooting section in the documentation.

1.5 Contributing

Your help is greatly appreciated! Suggestions, bug reports and feature requests are best opened as github issues. You could also first discuss in the gitter community. If you want to code something yourself, you are very welcome to submit a pull request!

1.6 License & Disclaimer

This software is licenced under the MIT license and (despite best testing efforts) comes **without any warranty**. The logo is inspired by the Anki logo (license) and the logo of the pandas package (license2). This library and its author(s) are not affiliated/associated with the main Anki or pandas project in any way.

CHAPTER 2

Troubleshooting

2.1 Getting help

Submit an issue on github or write in the gitter community. Thank you for improving this toolkit with me!

2.2 Common problems

- **Locked database**: While Anki is running, your database will be locked and you might not be able to access it. Simply close Anki and try again. Similarly Anki might refuse to open the database if `ankipandas` has currently opened it (be it in a Jupyter notebook or in a currently running project).

Note: Any unlisted problem that you ran into (and solved)? Help others by bringing it to my attention.

2.3 Debugging

For better debugging, you can increase the log level of `ankipandas`:

```
ankipandas.set_log_level("debug")
```

CHAPTER 3

Analysis

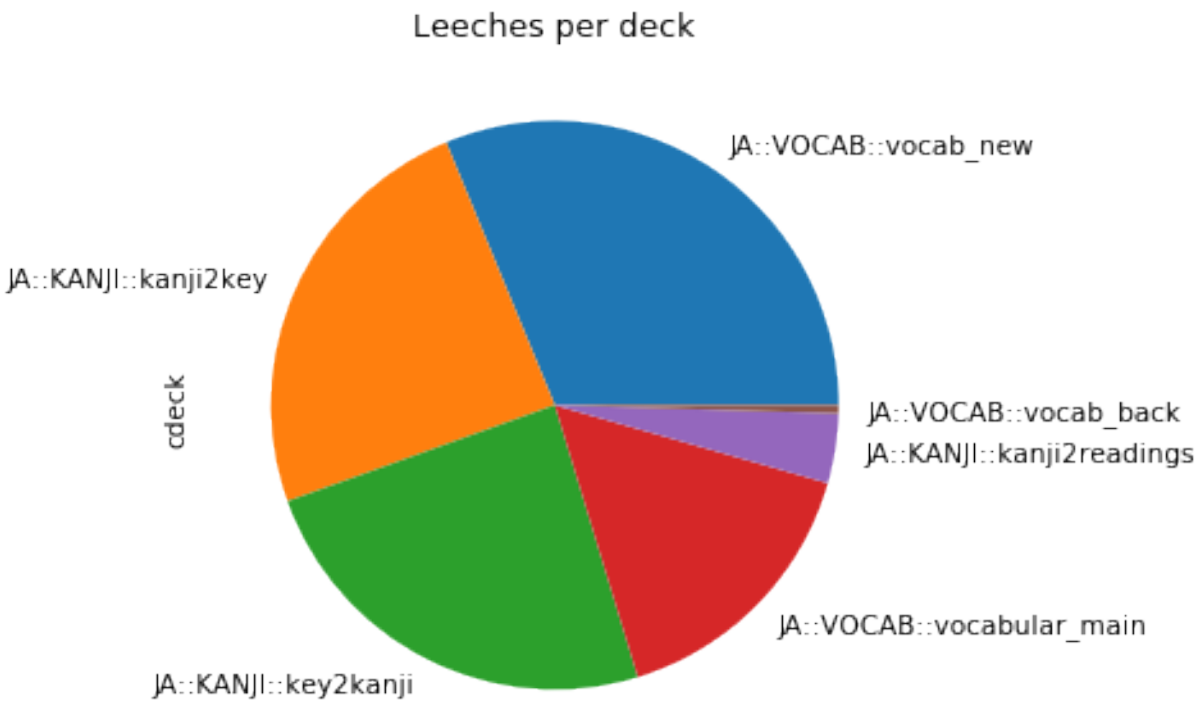
Note: All examples assume the line

```
col = Collection()
```

Or `col = Collection("/path/to/col.anki2")`, etc.

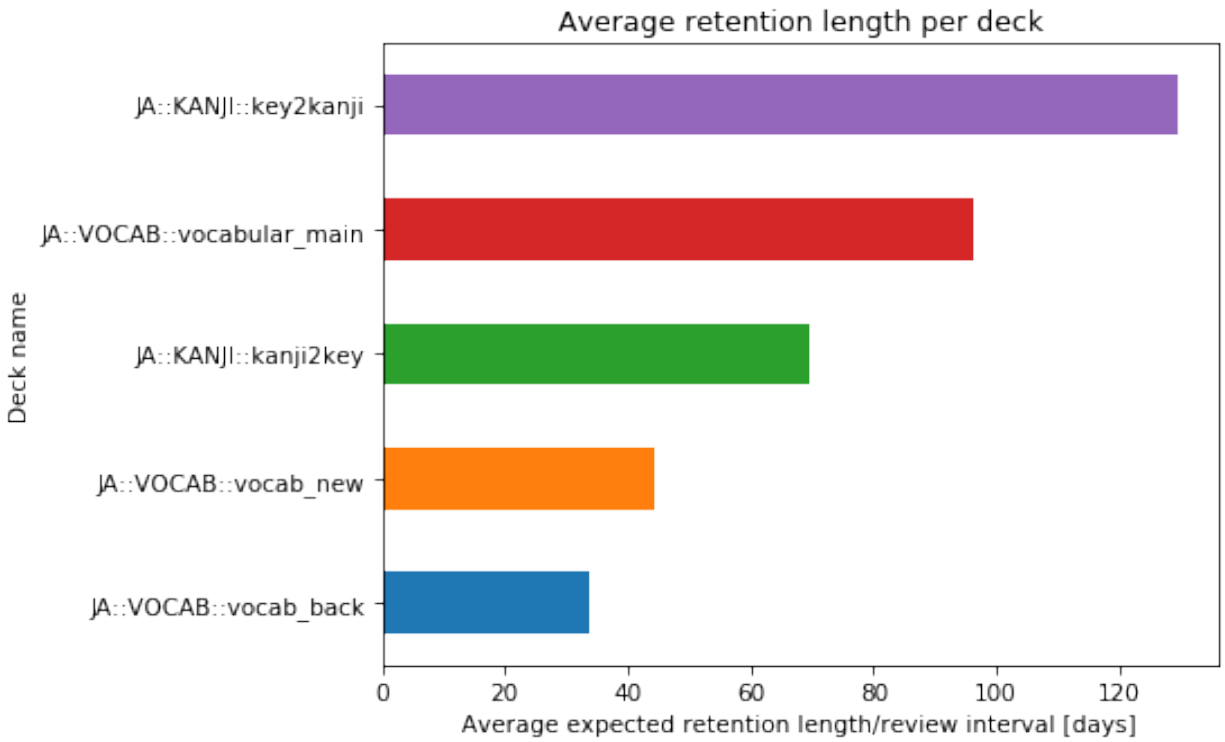
3.1 In which deck are the most leeches?

```
cards = col.cards.merge_notes()
counts = cards[cards.has_tag("leech")]["cdeck"].value_counts()
counts.plot.pie(title="Leeches per deck")
```



3.2 Which deck has the longest average retention rates?

```
grouped = col.cards.groupby("cdeck")
data = grouped.mean()["civl"].sort_values().tail()
ax = data.plot.barh()
ax.set_ylabel("Deck name")
ax.set_xlabel("Average expected retention length/review interval [days]")
ax.set_title("Average retention length per deck")
```



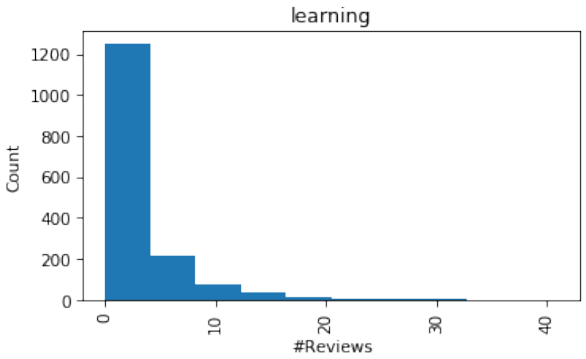
3.3 Repetitions vs type

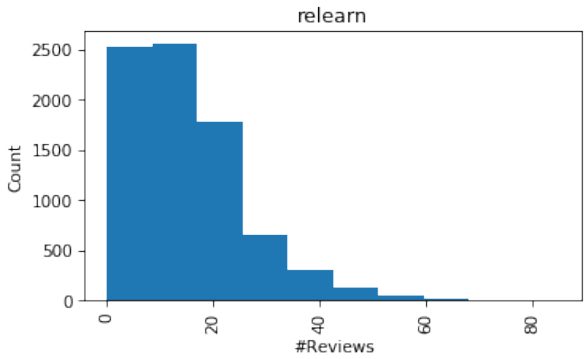
Minimal:

```
col.cards.hist("crepts", by="ctype")
```

Prettier:

```
axs = col.cards.hist(column="creps", by="ctype", layout=(1, 2), figsize=(12, 3))
for ax in axs:
    ax.set_xlabel("#Reviews")
    ax.set_ylabel("Count")
```





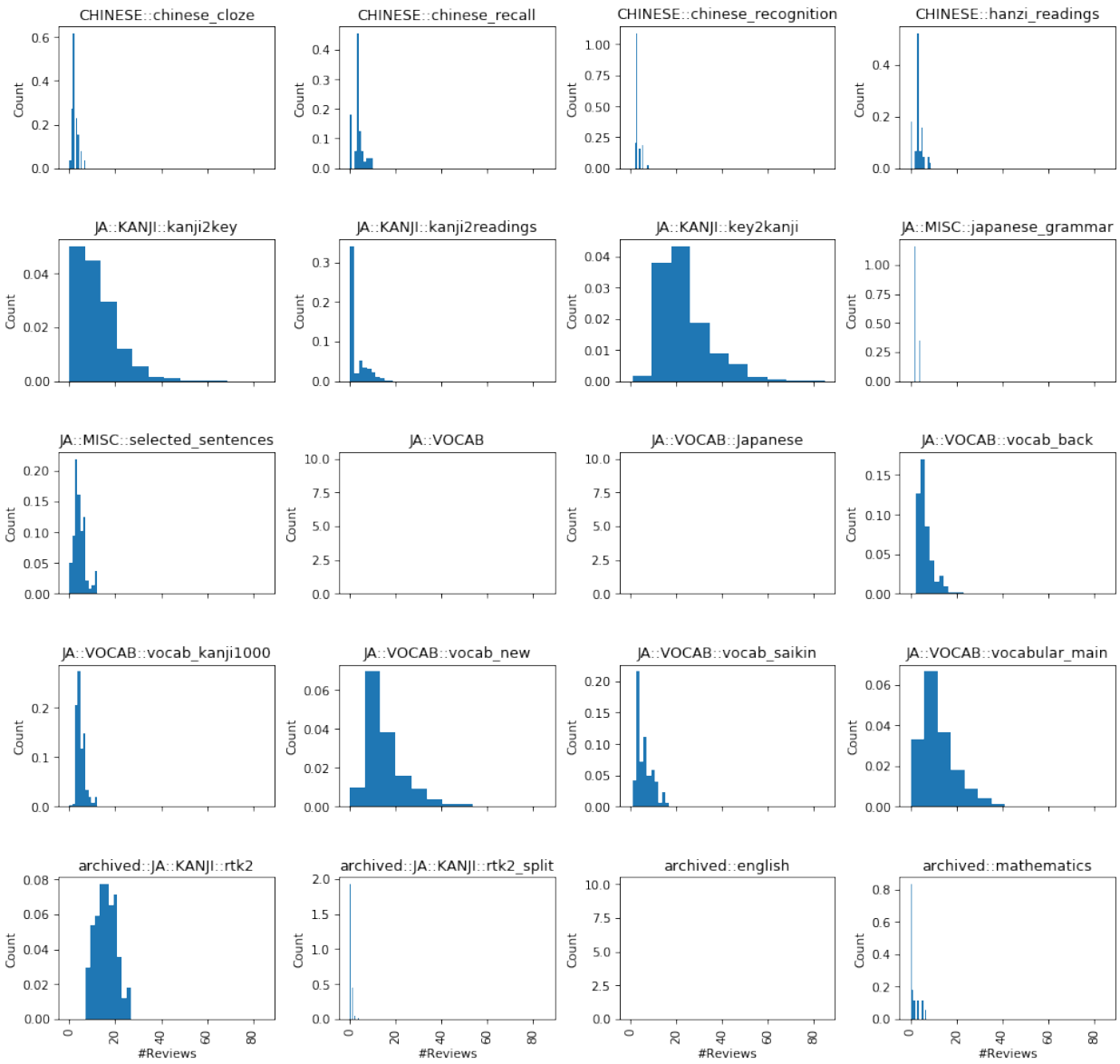
3.4 Repetitions vs deck

One liner:

```
col.cards.hist(column="creps", by="cdeck")
```

Prettier:

```
interesting_decks = list(col.cards.cdeck.unique())
interesting_decks.remove("archived::physics")
selected = col.cards[col.cards.cdeck.isin(interesting_decks)]
axss = selected.hist(
    column="creps",
    by="cdeck",
    sharex=True,
    layout=(5, 4),
    figsize=(15, 15),
    density=True,
)
for axs in axss:
    for ax in axs:
        ax.set_xlabel("#Reviews")
        ax.set_ylabel("Count")
```



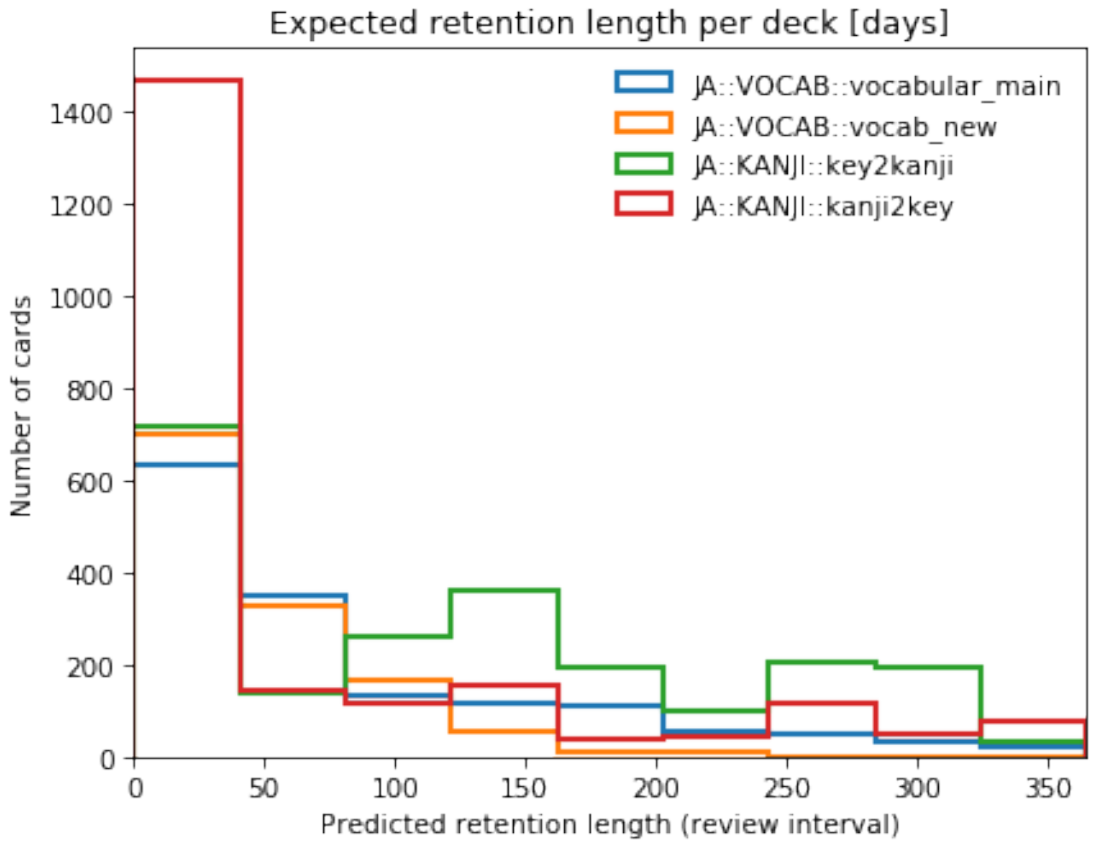
3.5 Retention distribution vs deck

```
import matplotlib.pyplot as plt
import numpy as np

ax = plt.gca()
for deck in col.cards.cdeck.unique():
    selected = col.cards[col.cards.cdeck == deck]["civl"]
    if len(selected) < 1000:
        continue
    selected.plot.hist(
        ax=ax,
        label=deck,
        histtype="step",
```

(continues on next page)

(continued from previous page)

```
        linewidth=2,
        xlim=(0, 365),
        bins=np.linspace(0, 365, 10),
    )
ax.set_xlabel("Predicted retention length (review interval)")
ax.set_ylabel("Number of cards")
ax.set_title("Expected retention length per deck [days]")
ax.legend(frameon=False)
```



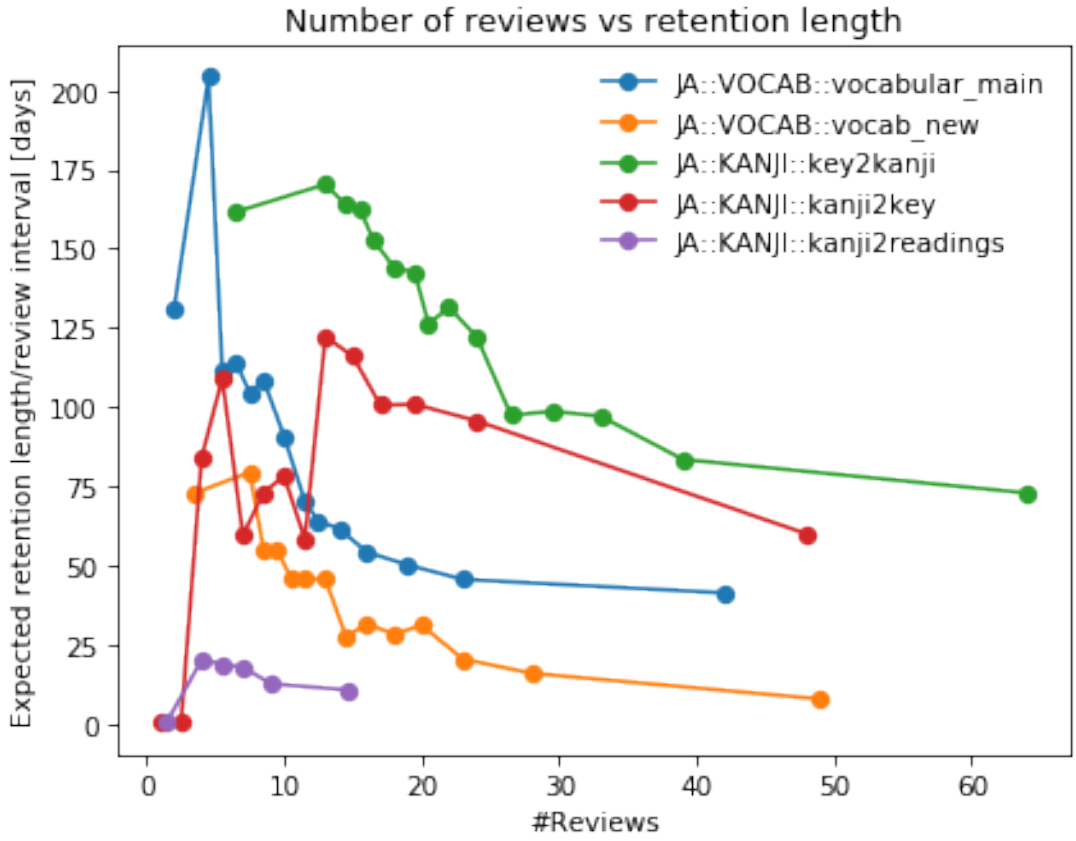
3.6 Reviews vs retention length vs deck

```
import pandas as pd

xs = []
ys = []
decks = []
for deck in col.cards.cdeck.unique():
    selected = col.cards[col.cards["cdeck"] == deck]
    if len(selected) < 500:
        continue
    decks.append(deck)
    binned = pd.qcut(selected["creps"], 15, duplicates="drop")
    results = selected.groupby(binned)["civl"].mean()
```

(continues on next page)

(continued from previous page)

```
    y = results.tolist()
    x = results.index.map(lambda x: x.mid).tolist()
    xs.append(x)
    ys.append(y)

ax = plt.gca()
for i in range(len(xs)):
    ax.plot(xs[i], ys[i], "o-", label=decks[i])
ax.set_xlabel("#Reviews")
ax.set_ylabel("Expected retention length/review interval [days]")
ax.set_title("Number of reviews vs retention length")
ax.legend(frameon=False)
```



CHAPTER 4

Collection

This is the starting point for most end-users. The *Collection* class loads the Anki collection and provides access to its notes, cards and reviews as pandas DataFrame objects.

class ankipandas.collection.**Collection**(*path=None*, *user=None*)

Bases: object

__init__(*path=None*, *user=None*)

Initialize *Collection* object.

Parameters

- **path** – (Search) path to database. See *db_path_input()* for more information.
- **user** – Anki user name. See *db_path_input()* for more information.

Examples:

```
from ankipandas import Collection

# Let ankipandas find the db for you
col = Collection()

# Let ankipandas find the db for this user (important if you have
# more than one user account in Anki)
col = Collection(user="User 1")

# Specify full path to Anki's database
col = Collection("/full/path/to/collection.anki2")

# Speficy partial path to Anki's database and specify user
col = Collection("/partial/path/to/collection", user="User 1")
```

path = None

Path to currently loaded database

db = None

Opened Anki database (sqlite3.Connection)

notes
Notes as *ankipandas.ankidf.AnkiDataFrame*.

cards
Cards as *ankipandas.ankidf.AnkiDataFrame*.

revs
Reviews as *ankipandas.ankidf.AnkiDataFrame*.

empty_notes()
Similar *ankipandas.ankidf.AnkiDataFrame* to *notes*, but without any rows.

empty_cards()
Similar *ankipandas.ankidf.AnkiDataFrame* to *cards*, but without any rows.

empty_revs()
Similar *ankipandas.ankidf.AnkiDataFrame* to *revs*, but without any rows.

summarize_changes(*output='print'*) → Optional[Dict[str, dict]]
Summarize changes that were made with respect to the table as loaded from the database. If notes/cards/etc. were not loaded at all (and hence also definitely not modified), they do not appear in the output.

Parameters **output** – Output mode: 'print' (default: print) or 'dict' (return as dictionary of dictionaries of format `{<type (cards/notes/...)>: {<key>: <value>}}`.

Returns None or dictionary of dictionaries

write(*modify=False*, *add=False*, *delete=False*, *backup_folder: Union[pathlib.PurePath, str] = None*)
Creates a backup of the database and then writes back the new data.

Danger: The switches `modify`, `add` and `delete` will run additional cross-checks, but do not rely on them to 100%!

Warning: It is recommended to run *summarize_changes()* before to check whether the changes match your expectation.

Note: Please make sure to thoroughly check your collection in Anki after every write process!

Parameters

- **modify** – Allow modification of existing items (notes, cards, etc.)
- **add** – Allow adding of new items (notes, cards, etc.)
- **delete** – Allow deletion of items (notes, cards, etc.)
- **backup_folder** – Path to backup folder. If None is given, the backup is created in the Anki backup directory (if found).

Returns None

CHAPTER 5

AnkiDataFrame

The class *AnkiDataFrame* is the central data structure in which we provide the notes, cards and review tables. Access it via an instance of *Collection*.

Example:

```
from ankipandas import Collection
col = Collection()

col.notes  # Notes as AnkiDataFrame
col.cards  # Cards as AnkiDataFrame
col.revs   # Reviews as AnkiDataFrame
```

class ankipandas.ankidf.**AnkiDataFrame**(**args*, ***kwargs*)

Bases: pandas.core.frame.DataFrame

__init__(**args*, ***kwargs*)

Initializes a blank *AnkiDataFrame*.

> **Warning:** It is recommended to directly initialize this class with the notes, cards or revs table, using one of the methods `notes()`, `cards()` or `revs()` instead!

Parameters

- ***args** – Internal use only. See arguments of pandas.DataFrame.
- ****kwargs** – Internal use only. See arguments of pandas.DataFrame.

fields_as_columns_prefix = None

Prefix for fields as columns. Default is `nfld_`.

classmethod init_with_table(*col*, *table*, *empty=False*)

db

Opened Anki database (sqlite3.Connection)

id
Return note/card/review ID as `pandas.Series` of integers.

nid
Note ID as `pandas.Series` of integers.

cid
Card ID as `pandas.Series` of integers.

rid
Review ID as `pandas.Series` of integers.

mid
Model ID as `pandas.Series` of integers.

did
Deck ID as `pandas.Series` of integers.

odid
Original deck ID for cards in filtered deck as `pandas.Series` of integers.

merge_notes(*inplace=False*, *columns=None*, *drop_columns=None*, *prepend='n'*, *prepend_clash_only=True*)
Merge note table into existing dataframe.

Parameters

- **inplace** – If False, return new dataframe, else update old one
- **columns** – Columns to merge
- **drop_columns** – Columns to ignore when merging
- **prepend** – Prepend this string to fields from note table
- **prepend_clash_only** – Only prepend the `prepend` string when column names would otherwise clash.

Returns New *AnkiDataFrame* if inplace==True, else None

merge_cards(*inplace=False*, *columns=None*, *drop_columns=None*, *prepend='c'*, *prepend_clash_only=True*)
Merges information from the card table into the current dataframe.

Parameters

- **inplace** – If False, return new dataframe, else update old one
- **columns** – Columns to merge
- **drop_columns** – Columns to ignore when merging
- **prepend** – Prepend this string to fields from card table
- **prepend_clash_only** – Only prepend the `prepend` string when column names would otherwise clash.

Returns New *AnkiDataFrame* if inplace==True, else None

fields_as_columns(*inplace=False*, *force=False*)
In the 'notes' table, the field contents of the notes is contained in one column ('flds') by default. With this method, this column can be split up into a new column for every field.

Parameters

- **inplace** – If False, return new dataframe, else update old one

- **force** – Internal use

Returns New `pandas.DataFrame` if inplace==True, else None

fields_as_list(*inplace=False*, *force=False*)
This reverts *fields_as_columns()*, all columns that represented field contents are now merged into one column 'nflds'.

Parameters

- **inplace** – If False, return new dataframe, else update old one
- **force** – Internal use

Returns New *AnkiDataFrame* if inplace==True, else None

list_tags() → List[str]
Return sorted list of all tags in the current table.

list_decks() → List[str]
Return sorted list of deck names in the current table.

list_models()
Return sorted list of model names in the current table.

has_tag(*tags=None*)
Checks whether row has a certain tag ('ntags' column).

Parameters **tags** – String or list thereof. In the latter case, True is returned if the row contains any of the specified tags. If None (default), True is returned if the row has any tag at all.

Returns Boolean `pd.Series`

Examples

```
# Get all tagged notes:
notes[notes.has_tag()]
# Get all untagged notes:
notes[~notes.has_tag()]
# Get all notes tagged Japanese:
japanese_notes = notes[notes.has_tag("Japanese")]
# Get all notes tagged either Japanese or Chinese:
asian_notes = notes[notes.has_tag(["Japanese", "Chinese"])]
```

has_tags(*tags=None*)
Checks whether row contains at least the supplied tags.

Parameters **tags** – String or list thereof. If None (default), True is returned if the row has any tag at all.

Returns Boolean `pd.Series`

Examples

```
# Get all notes tagged BOTH Japanese or Chinese
bilingual_notes = notes[notes.has_tags(["Japanese", "Chinese"])]
# Note the difference to
asian_notes = notes[notes.has_tag(["Japanese", "Chinese"])]
```

add_tag(*tags*, *inplace=False*)
Adds tag ('ntags' column).

Parameters

- **tags** – String or list thereof.
- **inplace** – If False, return new dataframe, else update old one

Returns New *AnkiDataFrame* if inplace==True, else None

remove_tag(*tags*, *inplace=False*)
Removes tag ('ntags' column).

Parameters

- **tags** – String or list thereof. If None, all tags are removed.
- **inplace** – If False, return new dataframe, else update old one

Returns New *AnkiDataFrame* if inplace==True, else None

was_modified(*other: pandas.core.frame.DataFrame = None*, *na=True*, *_force=False*)
Compare with original table, show which rows have changed. Will only compare columns existing in both dataframes.

Parameters

- **other** – Compare with this pandas.DataFrame. If None (default), use original unmodified dataframe as reloaded from the database.
- **na** – Value for new or deleted columns
- **_force** – internal use

Returns Boolean value for each row, showing if it was modified.

modified_columns(*other: pandas.core.frame.DataFrame = None*, *_force=False*, *only=True*)
Compare with original table, show which columns in which rows were modified.

Parameters

- **other** – Compare with this pandas.DataFrame. If None (default), use original unmodified dataframe as reloaded from the database.
- **only** – Only show rows where at least one column is changed.
- **_force** – internal use

Returns Boolean value for each row, showing if it was modified. New rows are considered to be modified as well.

was_added(*other: pandas.core.frame.DataFrame = None*, *_force=False*)
Compare with original table, show which rows were added.

Parameters

- **other** – Compare with this pandas.DataFrame. If None (default), use original unmodified dataframe as reloaded from the database.
- **_force** – internal use

Returns Boolean value for each row, showing if it was modified. New rows are considered to be modified as well.

was_deleted(*other: pandas.core.frame.DataFrame = None*, *_force=False*) → List
Compare with original table, return deleted indizes.

Parameters

- **other** – Compare with this pandas.DataFrame. If None (default), use original unmodified dataframe as reloaded from the database.
- **_force** – internal use

Returns Sorted list of indizes.

normalize(*inplace=False*, *force=False*)

Bring a *AnkiDataFrame* from the raw format (i.e. the exact format that Anki uses in its internal representation) to our convenient format.

Parameters

- **inplace** – If False, return new dataframe, else update old one
- **force** – If a previous conversion fails, *normalize()* will refuse to attempt another one by default. Use this option to force it to attempt in anyway.

Returns New *AnkiDataFrame* if inplace==True, else None

raw(*inplace=False*, *force=False*)

Bring a *AnkiDataFrame* into the raw format (i.e. the exact format that Anki uses in its internal representation) .

Parameters

- **inplace** – If False, return new dataframe, else update old one
- **force** – If a previous conversion fails, *raw()* will refuse to attempt another one by default. Use this option to force it to attempt in anyway.

Returns New *AnkiDataFrame* if inplace==True, else None

summarize_changes(*output='print'*) → Optional[dict]

Summarize changes that were made with respect to the table as loaded from the database.

Parameters **output** – Output mode: 'print' (default: print) or 'dict' (return as dictionary)

Returns None or dictionary

add_card(*nid: int, cdeck: str, cord: Union[int, List[int], None] = None, cmod: Optional[int] = None, cusn: Optional[int] = None, cqueue: Optional[str] = None, ctype: Optional[str] = None, civl: Optional[int] = None, cfactor: Optional[int] = None, creps: Optional[int] = None, clapses: Optional[int] = None, cleft: Optional[int] = None, cdue: Optional[int] = None, inplace=False*)

Similar to ankipandas.ankipdf.AnkiDataFrame.add_cards()

Parameters

- **nid** –
- **cdeck** –
- **cord** –
- **cmod** –
- **cusn** –
- **cqueue** –
- **ctype** –
- **civl** –
- **cfactor** –

- **creps** –
- **clapses** –
- **cleft** –
- **cdue** –
- **inplace** –

Returns:

add_cards(*nid: List[int], cdeck: Union[str, List[str]], cord: Union[int, List[int], None] = None, cmod: Union[int, List[int], None] = None, cusn: Union[int, List[int], None] = None, cqueue: Union[str, List[str], None] = None, ctype: Union[str, List[str], None] = None, civl: Union[int, List[int], None] = None, cfactor: Union[int, List[int], None] = None, creps: Union[int, List[int], None] = None, clapses: Union[int, List[int], None] = None, cleft: Union[int, List[int], None] = None, cdue: Union[int, List[int], None] = None, inplace=False*)

Add cards belonging to notes of one model.

Parameters

- **nid** – Note IDs of the notes that you want to add cards for
- **cdeck** – Name of deck to add cards to as string or list of strings (different deck for each nid).
- **cord** – Number of the template to add cards for as int or list thereof. The template corresponds to the reviewing direction. If left `None` (default), cards for all templates will be added. It is not possible to specify different cord for different nids!
- **cmod** – List of modification timestamps. Will be set automatically if `None` (default) and it is discouraged to set your own.
- **cusn** – List of Update Sequence Numbers. Will be set automatically (to -1, i.e. needs update) if `None` (default) and it is very discouraged to set your own.
- **cqueue** – 'sched buried', 'user buried', 'suspended', 'new', 'learning', 'due', 'in learning' (learning but next rev at least a day after the previous review). If `None` (default), 'new' is chosen for all cards. Specify as string or list thereof.
- **ctype** – List of card types ('learning', 'review', 'relearn', 'cram'). If `None` (default) 'learning' is chosen for all.
- **civl** – The new interval that the card was pushed to after the review. Positive values are in days, negative values are in seconds (for learning cards). If `None` (default) 0 is chosen for all cards.
- **cfactor** – The new ease factor of the card in permille. If `None` (default) 0 is chosen for all.
- **creps** – Number of reviews. If `None` (default), 0 is chosen for all cards.
- **clapses** – The number of times the card went from a 'was answered correctly' to 'was answered incorrectly'. If `None` (default), 0 is chosen for all cards.
- **cleft** – Of the form `a*1000+b`, with: `b` the number of reps left till graduation and `a` the number of reps left today. If `None` (default), 0 is chosen for all cards.
- **cdue** – Due is used differently for different card types: new: note id or random int, due: integer day, relative to the collection's creation time, learning: integer timestamp. If `None` (default), check that we're adding a new card and set to note ID.

- **inplace** – If `False` (default), return a new `AnkiDataFrame`, if True, modify in place and return new card IDs

Returns `AnkiDataFrame` if `inplace==True`, else list of new card IDs

add_notes(*nmodel: str, nflds: Union[List[List[str]], Dict[str, List[str]]], ntags: List[List[str]] = None, nid=None, nguid=None, nmod=None, nusn=None, inplace=False*)

Add multiple new notes corresponding to one model.

Parameters

- **nmodel** – Name of the model (must exist already, check *list_models()* for a list of available models)
- **nflds** – Fields of the note either as list of lists, e.g. `[[field1_note1, ... fieldN_note1], ..., [field1_noteM, ... fieldN_noteM]]` or dictionary `{field name: [field_value1, ..., field_valueM]}` or list of dictionaries: `[{field_name: field_value for note 1}, ..., {field_name: field_value for note N}]`. If dictionaries are used: If fields are not present, they are filled with empty strings.
- **ntags** – Tags of the note as list of list of strings: `[[tag1_note1, tag2_note1, ... ], ... [tag_1_noteM, ...]]`. If `None`, no tags will be added.
- **nid** – List of note IDs. Will be set automatically if `None` (default) and it is discouraged to set your own.
- **nguid** – List of Globally Unique IDs. Will be set automatically if `None` (default), and it is discouraged to set your own.
- **nmod** – List of modification timestamps. Will be set automatically if `None` (default) and it is discouraged to set your own.
- **nusn** – List of Update Sequence Number. Will be set automatically (to -1, i.e. needs update) if `None` (default) and it is very discouraged to set your own.
- **inplace** – If `False` (default), return a new `AnkiDataFrame`, if True, modify in place and return new note ID

Returns `AnkiDataFrame` if `inplace==True`, else new note ID (int)

add_note(*nmodel: str, nflds: Union[List[str], Dict[str, str]], ntags=None, nid=None, nguid=None, nmod=None, nusn=-1, inplace=False*)

Add new note.

Note: For better performance it is advisable to use *add_notes()* when adding many notes.

Parameters

- **nmodel** – Name of the model (must exist already, check *list_models()* for a list of available models)
- **nflds** – Fields of the note either as list or as dictionary `{field name: field value}`. In the latter case, if fields are not present, they are filled with empty strings.
- **ntags** – Tags of the note as string or Iterable thereof. Defaults to no tags.
- **nid** – Note ID. Will be set automatically by default and it is discouraged to set your own. If you do so and it already exists, the existing note will be overwritten.
- **nguid** – Note Globally Unique ID. Will be set automatically by default, and it is discouraged to set your own.

- **nmod** – Modification timestamp. Will be set automatically by default and it is discouraged to set your own.
- **nusn** – Update sequence number. Will be set automatically (to -1, i.e. needs update) if `None` (default) and it is very discouraged to set your own.
- **inplace** – If False (default), return a new `ankipandas.AnkiDataFrame`, if True, modify in place and return new note ID

Returns `ankipandas.AnkiDataFrame` if `inplace==True`, else new note ID (`int`)

help_col(*column*, *ret=False*) → Optional[str]

Show description/help about a column. To get information about all columns, use the *help_cols()* method instead.

Parameters

- **column** – Name of the column
- **ret** – If True, return as string, rather than printing

help_cols(*column='auto'*, *table='all'*, *ankicolumn='all'*) → pandas.core.frame.DataFrame

Show information about the columns and their interpretations. To get information about a single column, please use *help_col()*.

Parameters

- **column** – Name of a field or column (as used by us) or list thereof. If 'auto' (default), all columns from the current dataframe will be shown. If 'all' no filtering based on the table will be performed
- **table** – Possible values: 'notes', 'cards', 'revlog' or list thereof. If 'all' no filtering based on the table will be performed
- **ankicolumn** – Name of a field or column (as used by Anki) or list thereof. If 'all' no filtering based on the table will be performed

Returns Pandas DataFrame with all matches.

> **Warning:** As there are problems with text wrapping in pandas DataFrame, this method might change or disappear in the future.

static help(*ret=False*) → Optional[str]

Display short help text.

Parameters **ret** – Return as string instead of printing it.

Returns string if ret==True, else None

CHAPTER 6

Paths and Searching

Convenience functions to find the database and other system locations without the user having to specify full paths.

ankipandas.paths.**find_db**
Find path to anki2 database.

Parameters

- **search_paths** – Search path as string or pathlib object or list/iterable thereof. If None, some search paths are set by default.
- **maxdepth** – Maximal search depth.
- **filename** – Filename of the collection (default: `collections.anki2`)
- **user** – Username to which the collection belongs. If None, search for databases of any user.
- **break_on_first** – Stop searching once a database is found. This is obviously faster, but you will not get any errors if there are multiple databases matching your criteria.

Raises *If none ore more than one result is found* – `ValueError`

Returns pathlib.Path to the anki2 database

ankipandas.paths.**db_path_input**
Helper function to interpret user input of path to database.

1. If no path is given, we search through some default locations
2. If path points to a file: Take that file
3. If path points to a directory: Search in that directory

Parameters

- **path** – Path to database or search path or None
- **user** – User name of anki collection or None

Returns Path to anki database as `pathlib.Path` object

Raises

- *If path does not exist* – `FileNotFoundError`
- *In various other cases* – `ValueError`

ankipandas.paths.**db_backup_file_name**() → str
Time based file name of the backup file.

ankipandas.paths.**get_anki_backup_folder**(*path: Union[str, pathlib.PurePath], nexist='raise'*) → pathlib.Path
Return path to Anki backup folder.

Parameters

- **path** – Path to Aki database as `pathlib.Path`
- **nexist** – What to do if backup folder doesn't seem to exist: `raise` or `ignore`.

Returns Path to Anki backup folder as `pathlib.Path`.

ankipandas.paths.**backup_db**(*db_path: Union[str, pathlib.PurePath], backup_folder: Union[str, pathlib.PurePath] = None*) → pathlib.Path
Back up database file.

Parameters

- **db_path** – Path to database
- **backup_folder** – Path to backup folder. If None is given, the backup is created in the Anki backup directory.

Returns Path to newly created backup file as `pathlib.Path`.

CHAPTER 7

Raw Dataframes

These function implement the more direct interactions with the Anki database and provide basic functionality that is then used to implement the functionality in *Collection*, *ankipandas.ankidf.AnkiDataFrame* etc.

Warning: Please only use these function if you know what you are doing, as they come with less consistency checks as the functionality implemented in *Collection* and *ankipandas.ankidf.AnkiDataFrame*. Also note that the functions here are considered to be internal, i.e. might change without prior notice.

ankipandas.raw.**load_db**(*path: Union[str, pathlib.PurePath]*) → sqlite3.Connection
Load database from path.

Parameters **path** – String or pathlib.PurePath.

Returns sqlite3.Connection

ankipandas.raw.**close_db**(*db: sqlite3.Connection*) → None
Close the database.

Parameters **db** – Database (sqlite3.Connection)

Returns None

ankipandas.raw.**get_table**(*db: sqlite3.Connection*, *table: str*) → pandas.core.frame.DataFrame
Get raw table from the Anki database.

Parameters

- **db** – Database (sqlite3.Connection)
- **table** – cards, notes or revs

Returns pandas.DataFrame

ankipandas.raw.**get_empty_table**(*table: str*) → pandas.core.frame.DataFrame
Get empty table

Parameters **table** – cards, notes or revs

Returns class: *pandas.DataFrame*

ankipandas.raw.**get_info**
Get all other information from the database, e.g. information about models, decks etc.

Parameters **db** – Database (sqlite3.Connection)

Returns Nested dictionary.

ankipandas.raw.**set_table**(*db: sqlite3.Connection*, *df: pandas.core.frame.DataFrame*, *table: str*, *mode: str*, *id_column='id'*) → None
Write table back to database.

Parameters

- **db** – Database (sqlite3.Connection)
- **df** – The pandas.DataFrame to write
- **table** – Table to write to: 'notes', 'cards', 'revs'
- **mode** – 'update': Update all existing entries, 'append': Only append new entries, but do not modify, 'replace': Append, modify and delete
- **id_column** – Column with ID

Returns None

class ankipandas.raw.**NumpyJSONEncoder**(*skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *sort_keys=False*, *indent=None*, *separators=None*, *default=None*)
Bases: json.encoder.JSONEncoder

JSON Encoder that support numpy datatypes by converting them to built in datatypes.

default(*obj*)

ankipandas.raw.**set_info**(*db: sqlite3.Connection*, *info: dict*) → None
Write back extra info to database

Parameters

- **db** – Database (sqlite3.Connection)
- **info** – Output of *get_info()*

Returns None

ankipandas.raw.**get_ids**
Get list of IDs, e.g. note IDs etc.

Parameters

- **db** – Database (sqlite3.Connection)
- **table** – 'revs', 'cards', 'notes'

Returns Nested dictionary

ankipandas.raw.**get_deck_info**
Get information about decks.

Parameters **db** – Database (sqlite3.Connection)

Returns Nested dictionary

ankipandas.raw.**get_did2deck**
Mapping of deck IDs (did) to deck names.

Parameters **db** – Database (sqlite3.Connection)

Returns Dictionary mapping

ankipandas.raw.**get_deck2did**
Mapping of deck names to deck IDs

Parameters **db** – Database (sqlite3.Connection)

Returns Dictionary mapping of deck id to deck name

ankipandas.raw.**get_model_info**
Get information about models.

Parameters **db** – Database (sqlite3.Connection)

Returns Nested dictionary

ankipandas.raw.**get_mid2model**
Mapping of model IDs (mid) to model names.

Parameters **db** – Database (sqlite3.Connection)

Returns Dictionary mapping

ankipandas.raw.**get_model2mid**
Mapping of model name to model ID (mid)

Parameters **db** – Database (sqlite3.Connection)

Returns Dictionary mapping

ankipandas.raw.**get_mid2sortfield**
Mapping of model ID to index of sort field.

ankipandas.raw.**get_mid2fields**
Get mapping of model ID to field names.

Parameters **db** – Databse (sqlite3.Connection)

Returns Dictionary mapping of model ID (mid) to list of field names.

ankipandas.raw.**get_cid2nid**
Mapping card ID to note ID.

Parameters **db** – Database (sqlite3.Connection)

Returns Dictionary

ankipandas.raw.**get_cid2did**
Mapping card ID to deck ID.

Parameters **db** – Database (sqlite3.Connection)

Returns Dictionary

ankipandas.raw.**get_nid2mid**
Mapping note ID to model ID.

Parameters **db** – Database (sqlite3.Connection)

Returns Dictionary

CHAPTER 8

Utilities

Various utilities of this package.

Warning: These utilties are less aimed at end users and might therefore be subject to change.

ankipandas.util.log.**get_logger**()
Sets up global logger.

ankipandas.util.log.**set_log_level**(*level: Union[str, int]*) → None
Set global log level.

Parameters **level** – Either an int (https://docs.python.org/3/library/logging.html#levels) or one of the keywords, 'critical' (only the most terrifying of log messages), 'error', 'warning', 'info', 'debug' (all log messages)

Returns None

DataFrame utilities.

ankipandas.util.dataframe.**replace_df_inplace**(*df: pandas.core.frame.DataFrame*, *df_new: pandas.core.frame.DataFrame*) → None
Replace dataframe 'in place'.

Parameters

- **df** – pandas.DataFrame to be replaced
- **df_new** – pandas.DataFrame to replace the previous one

Returns None

ankipandas.util.dataframe.**merge_dfs**(*df: pandas.core.frame.DataFrame*, *df_add: pandas.core.frame.DataFrame*, *id_df: str*, *inplace=False*, *id_add='id'*, *prepend=''*, *replace=False*, *prepend_clash_only=True*, *columns=None*, *drop_columns=None*)
Merge information from two dataframes.

Parameters

- **df** – Original pandas.DataFrame
- **df_add** – pandas.DataFrame to be merged with original pandas.DataFrame
- **id_df** – Column of original dataframe that contains the id along which we merge.
- **inplace** – If False, return new dataframe, else update old one
- **id_add** – Column of the new dataframe that contains the id along which we merge
- **prepend** – Prepend a string to the column names from the new dataframe
- **replace** – Replace columns
- **prepend_clash_only** – Only prepend string to the column names from the new dataframe if there is a name clash.
- **columns** – Keep only these columns
- **drop_columns** – Drop these columns

Returns New merged pandas.DataFrame

ankipandas.util.misc.**invert_dict**(*dct: dict*) → dict

Invert dictionary, i.e. reverse keys and values.

Parameters **dct** – Dictionary

Returns Dictionary with reversed keys and values.

Raises ValueError if values are not unique.

ankipandas.util.misc.**flatten_list_list**(*lst: List[List[Any]]*) → List[Any]

Takes a list of lists and returns a list of all elements.

Parameters **lst** – List of Lists

Returns list

ankipandas.util.checksum.**field_checksum**(*data: str*) → int

32 bit unsigned number from first 8 digits of sha1 hash. Apply this to the first field to the the field checksum that is used by Anki to detect duplicates.

Parameters **data** – string like

Returns int

Python Module Index

a

Index

Symbols

A

B

C

D

E

F

G

H

I

L

M

N

O

P

R

S

W