OzCore

Release 1.2

Ozgur Kalan

Apr 06, 2021

CONTENTS

Helper functions.

Functions can be directly called from core

usage example:

```
from ozcore import core

core.translate("merhaba")
```

ozcore.core.helpers.**now_prefix**(*separator='-'*, *format='now'*)
datetime today or now as prefix

Parameters

- **separator** (*str*) – default None, a seperator string for date and time
- **format** (*str*) – default Now,

```
("now")=> "%y%m%d-%H%M%S"
("today")=> "%y%m%d"
("or any valid format")=> "%y%m%d-%H%M%S"
```

Returns str

Hint: useful for naming files

---

ozcore.core.helpers.**clean_html**(*html*)
cleans given html and returns a markdown

Parameters **html** – html markup str as input

Returns str, as mardown

ozcore.core.helpers.**md_2_html**(*md_text*)
converts markdown to html

Parameters **md_text** – markdown str as input

Returns str, as html markup

ozcore.core.helpers.**translate**(*text='None'*, *dest='en'*, *src='auto'*, *html=False*)

translate text via Google Translator.

connects free to google translator,
limits text to max. 12.000 chars

Parameters

- **text** – str, text to translate (may use html or markdown too)
- **dest** – str, destination lang, defaults to english (en)
- **src** – str, source lang, defaults to auto
- **html** – bool, if cleaned and set False returns Markdown, if set True returns html

Returns str or html

Warning:

given text should not have special chars
and may need to be cleaned by _clean_html

ozcore.core.helpers.**serialize_a_json_field**(*val*, *node=None*)

safely eval a field with a string list or dict inherited from a json file e.g. [{name:test}] => list object having dict node 'name'

Parameters

- **val** – json | dict, field value passed
- **node** – str, key name in the dictionary

Returns

- semicolon seperated values if val is a set, dict or list
- if node is given, returns the values in the node as semicolon separated string
- if val is None, returns None
- if fails to the opretion returns back the val itself

Hint: useful in serializing fields in a dataframe having dict like objects

ozcore.core.helpers.**dirme**(*me*)

lists methods of a given module in Jupyter Notebook

Parameters **me** – str, module, argument, method

Returns

- displays module, arg or method as a pandas DataFrame
- display option, set to the len of df (in Jupyter Notebook)

ozcore.core.helpers.**unzip**(*url: str*, *dest: pathlib.PosixPath*, *chunk_size: int = 1048576*, *remove_zip: bool = False*)

Downloads and unzips a zip file

Parameters

- **url** – str, uri to zip file
- **dest** – PosixPath, destination folder
- **chunk_size** – int, default 1 MB
- **remove_zip** – bool, default False, unlinks zip file after unzip operation

Returns tqdm progress bar and typer echo messages

CHAPTER

ONE

JUPYTER NOTEBOOK SETUP

Jupyter Notebook helper functions mainly to setup on initial kernel start.

class ozcore.core.jupyter.**Jupyter**

Jupyter Notebook helper functions

run setup function in a jupyter notebook, preferably in an init file like *bot_init.ipynb*

usage:

```
%reset -f
from ozcore.core import jupyter
jupyter.Jupyter().setup()
```

setup()

setup Jupyter Notebook with beloved attributes

- large view
- pandas view options

large_view()

larger view for Jupyter Notebook

pandas_view_options()

pandas view options

CHAPTER

TWO

PATH AND FOLDER METHODS

Useful paths and folders, search in folders and temporary folder helper methods.

Warning: Folder paths are for **OSX** only.

Note: Special folders like Google Drive and OneDrive may not be available in your directories. Or your directory structure may be different.

class ozcore.core.path.folders.**Folder**

Essential folder paths as posixpath:

- Downloads
- gDrive
- iCloud
- OneDrive

Hint: Search for a folder with glob option

Apps folder in this package:

- folders in apps as posixpath enum
- list of folders in apps folder

Example, Search Downloads folder:

```
from ozcore import core
core.folder.search(path=core.folder.Downloads, file="the_file_searched.xl**")
```

Example, is_a_subfolder:

```
from ozcore import core
core.folder.is_a_subfolder(path_to_check=Path("."), parent_folder=core.BASE_DIR)
```

property Downloads

MacOS Downloads folder path

Returns POSIXPATH to Downloads folder

property gDrive
Google Drive folder path

Returns POSIXPATH to gDrive

property iCloud
iCloud folder path

Returns POSIXPATH to iCloud

property OneDrive
OneDrive folder path

Returns POSIXPATH to OneDrive

search(*path*, *file*)
search files in a given folder path

Parameters

- **path** – a str or posixpath path
- **file** – str, file to search; searches as glob, e.g. "/.json"**

Returns list, files in POSIXPATH type

usage:

```
from ozcore import core
core.folder.search(core.folder.gDrive.joinpath("folder name"), "*.xlsx")
```

is_a_subfolder(*path_to_check: pathlib.PosixPath*, *parent_folder: pathlib.PosixPath*)
check if a given path is in parent folder

Parameters

- **path_to_check** – posixpath, a path to match with base folder
- **parent_folder** – posixpath, the base aka parent folder

Returns boolean

check_path(*path: Union[str, pathlib.PosixPath]*, *is_file: bool = False*) → pathlib.PosixPath
Checks the given `path` param, converts it to an absolute path

Warning: All relative paths are ignored and Exception will be raised.

Note: This method checks the type of the arguments and raise Exception if not satisfied.

Parameters

- **path** – str|PosixPath, can be Pathlike path or a relative path
- **is_file** – bool, default False, False: a directory, True: a file

Returns Absolute PosixPath; checks if path exists.

CHAPTER

THREE

TMP FOLDERS AND FILES

Helper class for Temporary Folders and Files

class ozcore.core.path.tmp_folders.**TMP_Folder**(*tmp_root='.'*, *tmp_folder_name='tmp'*)

Helper class for Temporary Folders and Files

methods:

- tmp folder root
- tmp folder path
- cleans tmp folder files with given conditions

Hint: Useful for cleaning `tmp` folders espaecially when fetching csv files into `tmp` folders

---

usage:

```
from ozcore.core.path import TMP_Folder
tmp = TMP_folder(tmp_root="root/folder/path/", tmp_folder_name="tmp")

tmp.root
# ../../root_folder

tmp.path
# ../../root folder/tmp_folder_path

tmp.clean(glob="**/*.json", n=3)
# ... cleans all json files up to latest 3 in the tmp

tmp.clean(glob="*", n=-1)
# ... cleans all files in the tmp
```

property root

root folder path of the temporary folder as posixpath

Returns posixpath path to root of the tmp folder

property path

path of the tmp folder as posixpath

Returns posixpath to tmp folder

clean(*glob*, *n=5*, *reverse=False*)

deletes old files in the tmp folder keeps last n files (sorted asc or desc)

Parameters

- **n** – number of files to *keep*!, default:5; -1 for all files
- **glob** – search tmp folder to match files,
- **reverse** – default False if sort files asc; for desc order set to True

usage:

```
# e.g. **/*.json for all json files in recursive subfolders
tmp_folder.clean(glob="**/*.json", n=3)

# e.g. * for all files in the folder
tmp_folder.clean(glob="*", n=-1)
```

CHAPTER

FOUR

DUMMY RECORDS

Dummy records, ready to use

Hint: can be directly called from core as `core.dummy`

usage:

```
from ozcore import core

core.dummy.emp
#...returns a dataframe with 10 records

core.dummy.dataframe(n=3, template="emp", verbose=True)
#...returns a dataframe with 3 records using **emp** template

core.dummy.df1
#...returns a dataframe with faked with seed 99 having shape(5,5)

core.dummy.df2
#...returns another dataframe with faked with seed 99 having shape(5,5)

core.dummy.df1
#...returns another dataframe with faked with seed 99 having shape(5,4)

core.dummy.fake.name_female()
# ... you also have access to Faker class
```

Todo: New templates other than emp, e.g. np arrays shall be included

class ozcore.core.data.dummy.**Dummy**
ready-to-use dummy records, created using Faker

property emp
ready to use dummy employee dataframe with 10 records

Returns dataframe with 10 records

dataframe(*template='emp'*, *n=10*, *verbose=True*)
create a dummy dataframe

Parameters

- **n** – int, number of records, default 10

- **template** – str, default "emp", choose from ready templates
- **verbose** – bool, default True

Returns

- if verbose, returns the dataframe
- else, assigns class df values

> **Warning:** Cannot assign 1 record only, if n<2 then n=2

_json_employee(*n=10*)
dummy json object with keys for employee template

Parameters **n** – int, number of records, default 10

Returns json object

property df1
Dataframe with Faker's seed 99

Returns Dataframe shape(5,5), 5th col as datetime object

property df2
Dataframe with Faker's seed 99

Returns Dataframe shape(5,5), 5th col as dict object

property df3
Dataframe with Faker's seed 99

Returns Dataframe shape(5,4)

_make_df_w_seed_99(*df_no=1*)
creates a dummy Dataframe with Faker seed 99

Parameters **df_no** – int, default 1, can be 1 , 2 or 3

Returns

- a dataframe
- all three dataframes have their first 2 columns identical
- df_no=1: (5,5), 5th col as datetime object
- df_no=2: (5,5), 5th col as dict object
- df_no=3: (5,4)

CHAPTER

FIVE

DATAFRAME HELPER METHODS

Data and dataframe helper class for the module Core

Hint: can be directly called from core as `core.df`

basic usage:

```
from ozcore import core
core.df.search(df, q="something")
```

class ozcore.core.data.dataframe.**Dataframe**

helper methods for dataframe operations

update_a_df_column(*df_to_update: pandas.core.frame.DataFrame*, *df_as_source: pandas.core.frame.DataFrame*, *unique_col: str*, *col_to_update: str*)

Updates a Dataframe column with a source Dataframe based on their common unique columns

Parameters

- **df_to_update** – dataframe, main df to be updated
- **df_as_source** – dataframe, source df to update the main df
- **unique_col** – str, common columns (should have same name) to match records, this unique column must have unique values
- **col_to_update** – str, which column value to be updated

Returns a copy of the updated DataFrame

Warning: index is reset during the update

pngTable(*df: pandas.core.frame.DataFrame*, *colwidth_factor: float = 0.2*, *fontsize: int = 12*, *formatFloats: bool = True*, *save: bool = False*, *in_folder: Optional[pathlib.PosixPath] = None*)

Displays or saves a table as png. Uses matplotlib => pandas plotting table.

Parameters

- **df** – dataframe or pivot table
- **colwidth_factor** – float, default 0.20, defines the width of columns
- **fontsize** – int, default 12
- **formatFloats** – bool, default True, formats as two digit prettiy floats
- **save** – saves the png file as table.png

- **in_folder** – posixpath, default None, folder to save the png file

Returns png file in Downloads folder

compare_two_df(*df_1*, *df_2*, *col_to_compare: str*, *side='both'*)

Compares two dataframes based on a given column, aka given common Series

> **Warning:** This comparison is only checking the identical values in a Series. Other columns may not match.

Parameters

- **df_1** – dataframe 1
- **df_2** – dataframe 2
- **col_to_compare** (*str*) – column to make the comparison, which is common
- **side** – str, default *both*, options: *left*, *right*

Returns

- a dataframe with diffrences of df_1 from df_2
- empty if all match

add_a_col_from_a_df(*into_df: pandas.core.frame.DataFrame*, *from_df: pandas.core.frame.DataFrame*, *unique_col: str*, *col_to_add: str*)

Add a column into a dataframe from another dataframe

Parameters

- **into_df** – dataframe, main df, which will be updated with a new column
- **from_df** – dataframe, source df, which has the column to add into main df
- **unique_col** – str, column name which is common in both dataframes
- **col_to_add** – str, column to be added from source dataframe

Returns

- main dataframe filled with the new column and values, where unique column matches

> **Warning:** this method assumes no index

search(*df_to_search*, *q*, *columns=None*)

Search all or any column of a dataframe, where columns having str (type: object)

Parameters

- **df_to_search** – dataframe to be searched
- **q** – str, query term
- **columns** – str | list, default None, columns to search, if None all columns

Returns a dataframe with found records

Note: index columns are not included.

CHAPTER

SIX

CSV OPERATIONS HELPER CLASSES

In order to ease the CSV related operations, there are two helper classes to assist.

Module Core is directly calling the Base class. Process class is for manipulating CSV files and should be called explictly.

6.1 Base Class CSV

Base class for csv operations

class ozcore.core.data.csv.base.**Base**

Base class for csv operations.

Usage:

```
from ozcore import core

csv = core.csv

csv.read("path_to_csv_file", **kwargs)

csv.save(df_altered, index=False, verbose=False, overwrite=False, **kwargs)
# warning: this saves over the read csv file!

csv.search(q="something")
# searches every column

# but if certain columns desired to be searched than define it before action
csv.searchable = ["col1"]
csv.search(q="something in col1")
```

read(*path*, *verbose=True*, ***kwargs*)

read a csv file

Parameters

- **path** – str|posixpath, path to csv file
- **kwargs** – pandas read_csv arguments

Returns fills self.actual_df and self.df

save(*df_altered*, *index=False*, *verbose=False*, *overwrite=False*, ***kwargs*)

saves altered dataframe on the csv file

paramaters: df_altered: Dataframe index: bool, default False verbose: bool, default False overwrite: bool, default False kwargs: pandas to_csv arguments

Returns logging if verbose is True

Warning: if overwrite is True, dumps given df into file without checking its len with actual_df

search(*q*)
search records based on self._searchable argument (col name to search given in child classes)

Returns dataframe slice from search results

6.2 Process Class CSV

Processing and marking a csv file.

class ozcore.core.data.csv.process.**Process**(*path*)
Class for processing and marking csv files. Drived from Base class. ! And, cannot be initiated if validation fails.

Hint: This module can be used to modify fields in a csv file and tick mark completed rows...

Warning: Processing CSV assumes that the csv file has no index defined. Also, no kwargs are passed to read_csv method of Pandas.

Notes

You need to define a path to the csv file in order to initiate the Process class.

Warning: If no process header found in CSV, this class raises an Exception

usage:

```
from ozcore.core.data.csv.process import Process as csv_process

csv = csv_process(path="path_to_csv")

csv.focus(0)
# focus first record

csv.processed()
# mark first record as processed and save it to csv files

csv.next()
# focus on next unprocessed record (marked as 0) if exists
```

create a processed column:

```
# to use this class, a processed column with 0 values should be available
# warning: there may be similar column, please check before proceeding

from ozcore import core

df = core.csv.read(file)

df = df.assign(processed=0)
df.processed = df.processed.astype(int)
df.loc[:,"processed"] = 0

core.save(df, index=False, overwrite=True)

# now you can initiate the Process class
```

deleting the processed column:

```
# when you are finished with marking csv file you can remove it
# warning: be sure to remove

from ozcore import core

df = core.csv.read(file)

df.drop(columns="processed", inplace=True)

core.save(df, index=False, overwrite=True)

# column is removed
```

property validate_csv
Validates the CSV if fits the strict rules for processing a CSV file.

focus(*index*, *verbose=True*)

the record which is in progress all write actions is applied to this Serie

paramaters: index: int, index number of the record being focused verbose: bool, default True

Returns fills self.focus_index and displays the Series if verbose displays a warning msg

next()
next unprocessed item where processed==0 is displayed and assigned as focused

Returns Series

processed(*revert=False*)
marks a records as processed by assigning 1

Parameters **revert** – boolean, default False

usage: when a records focused by focus(index) or next() methods assigns 1 to processed column (0 if revert is True) saves the file

CHAPTER

SEVEN

SQLITE OPERATIONS HELPER CLASSES

SQLite operations are a bit tricky. In order to ease the SQLite related operations, there are two helper classes to assist.

Mainly, only `Sqlite` class in `sqlite` module is used. The module and class separation is made due to better code managament.

All methods can be directly called from `core`.

Example:

```
from ozcore import core

core.sql.read(...)
```

7.1 Sqlite Class

sqlite helper methods

class ozcore.core.data.sqlite.sqlite.**Sqlite**

Sqlite helper methods using `ORM` class

Set the engine before using or if .db is in the current folder, creates the engine automatically

> **Warning:** set_engine() before proceeding any other method!

usage:

```
from ozcore import core

core.sql.set_engine(engine=core.sql.engines.engine_name)

core.sql.tables.table_name
core.sql.path_to_database

core.sql.read(table_name, engine, limit=100)
```

create_engine(*path: Union[str, pathlib.PosixPath]*)

Creates an engine.

Parameters **path** – str | posixpath, path to the sqlite db

Returns Engine object

Note: Allowed extensions: “db”,”sqlite”,”sqlite3”

set_engine(*engine: Union[sqlalchemy.engine.base.Engine, str, pathlib.PosixPath]*, *return_engine: bool = False*)
Set the engine to work with.

Parameters

- **engine** – sqlalchemy engine, Posixpath, or str
- **return_engine** – bool, default False, returns the engine set

Returns

- fills self engine
- if returns=True, returns the engine set

Warning: To work with **core.sql** methods, you should first set the engine!

usage:

```
# with an engine name
core.sql.set_engine(engine=core.sql.engines._ENGINE_NAME)

# with an relative path
core.sql.set_engine(engine="./sample.db")
```

property metadata
returns sqlalchemy metadata of the current engine

property tables
enum tables in a database

Returns a table name from Enum also, assigns table names as a list in self.tables_list

usage:

```
core.sql.tables.table_name
```

columns(*table_name*)
enum columns in a table

Parameters **table_name** – str|enum

Returns enum columns (.name as str name, .value as sa col object) also assigns this query to self.columns_list as a dict

column_exists(*table_name*, *col_name*)
checks if a column name exists in a table

Parameters

- **table_name** – str|enum
- **col_name** – str|enum

Returns boolean

table_exists(*table_name*)
checks if a table name exists in a database

Parameters **table_name** – str|enum

Returns boolean

property path_to_database
path to database from current engine

Parameters **engine** –

read(*table_name*, *limit=None*, *index_column=None*)
read a table

Parameters

- **table_name** – str
- **limit** – int, default None

Returns a dataframe with table results

7.2 ORM Class

sqlite alter operations

class ozcore.core.data.sqlite.orm.**ORM**
Sqlite alter operations with ORM and Alembic

> **Warning:** The sa_ prefixed methods are for advanced usage

sa_add_a_column(*table_name*, *column_name*, *type_=<class 'sqlalchemy.sql.sqltypes.TEXT'>*)
alter table: add a new column in the given table

Parameters

- **table_name** – str
- **column_name** – str, name for the new column

Returns

- creates the column in the table as Text type
- returns Error if exception

> **Warning:** new column is created as sqlalchemy.sql.sqltypes.Text

sa_drop_a_column(*table_name*, *column_name*)
alter table: drop a column from a table

Parameters

- **table_name** – str
- **column_name** – str, as column name or a sqlalchemy Column object

Returns drops the column in the given table

sa_change_column_type(*table_name*, *column_name*, *type_*)
alter table: change a column's type

Parameters

- **table_name** – str
- **column_name** – str, as column name or a sqlalchemy Column object
- **type** – Sqlalchemy Type

Returns

- True if type is changed successfully

usage:

```
import core

core.sql.("table_name", "column_name", type_=sa.TEXT)
# type changed to SQLalchemy Text type
```

sa_table(*table_name: str*)
Sqalchemy Table object of a given table

Parameters **table_name** – str or Sqlalchemy Table object

Returns Sqlalchemy Table object of the set engine

sa_column(*table_name: str*, *column_name: str*)
Sqalchemy Coulumn object of a given table

Parameters

- **table_name** – str
- **column_name** – str

sa_update_a_record(*table_name*, *column_name*, *compare_column*, *compare_val*, *val*)
update a single record in a given column of a table

Parameters

- **table_name** – str or Sqlalchemy Table object
- **column_name** – str or Sqlalchemy Column object
- **compare_column** – str, the common unique columnn to compare record
- **compare_val** – mixed, a value to compare in compare_column
- **val** – mixed, the new value of the record

sa_update_a_column(*table_name*, *column_name*, *compare_column*, *source_df*)
update a column's records based on a given df_slice

Parameters

- **table_name** – str or Sqlalchemy Table object
- **column_name** – str or Sqlalchemy Column object
- **compare_column** – str, the common unique columnn to compare record
- **compare_val** – mixed, a value to compare in compare_column
- **source_df** – source records as a DataFrame or Series, to update the Table

Warning: index of source_df is ignored

CHAPTER

EIGHT

QGRID EXTENDED

Qgrid is a Jupyter notebook widget which uses `SlickGrid` to render pandas DataFrames within a Jupyter notebook. This allows you to explore your DataFrames with intuitive scrolling, sorting, and filtering controls, as well as edit your DataFrames by double clicking cells.

This extention includes helper classes to quickly integrate in the `Module Core`. Also, Apps can inherit the `Grid` abstract class to detail views in Jupyter Notebook.

8.1 Qgrid Usage

Basic usage is by calling from core:

```
from ozcore import core

core.gridq.view(df, handler=True, minor=None)
```

For consuming abstract class in Apps:

```
from ozcore.core.qgrid.grid import Grid

class MySampleClass:
    ....
    def grid(self):
        # call grid's inner class
        _grid = self._View()
        _grid.view(df_slice=self.df)

    class _View(Grid):
        # inner class to inherit core > qgrid > view method

        def __init__(self, cards=False):
            super().__init__()

        def _handler_view_selection_changed(self, event, qgrid_widget):
        # ....  override handler

        def edit()
        # .... override edit
```

8.2 Qgrid Class Methods

qgrid abstract class

class ozcore.core.qgrid.grid.**Grid**
abstract class for calling qgrid

inherits: Base class View class

*** view**
displays with settings in Base class

*** edit**
abstract method, to be implemented

helper methods:

_handler_view_selection_changed()
already implemented in View class, can be altered by re-defining

usage:

```
# simple usage:
from ozcore import core
core.gridq.view(dataframe)

# integrate with inner class:
from ozcore.core.qgrid_.grid import Grid

class MySampleClass:
    ....
    def grid(self):
        # call grid's inner class
        _grid = self._View()
        _grid.view(df_slice=self.df)

    class _View(Grid):
        # inner class to inherit core > qgrid > view method

        def __init__(self, cards=False):
            super().__init__()

        def _handler_view_selection_changed(self, event, qgrid_widget):
        # ....  override handler
        def edit()
        # .... override edit
```

__init__()
Initialize self. See help(type(self)) for accurate signature.

edit(*df_slice*, ***kwargs*)
edit qgrid fields

NotImplemented

_clean_qgrid_js()
Javascript injection for qgrid

Returns

- JS integrated in Jupyter Notebook for Qgrid
- removes default buttons,
- adds a js button,
- adds js to scroll action of qgrid

_display_qgrid(*q*)
build qgrid display for Jupyter Notebook

usage:

```
q = self._qgrid(df_slice=df_slice, **kwargs)
display_qgrid(q)

# define before any headers to receive qgrid info, e.g. shape
```

_handler_view_selection_changed(*event*, *qgrid_widget*)
event handler for view when qgrid on selection_changed

Returns displays selected row in footer single rec Output

_qgrid(*df_slice=None*, *minor=None*, ***kwargs*)
Custom qgrid wrapper

Parameters

- **df_slice** – Dataframe to be displayed
- **minor** – list, columns to display
- **kwargs** – arguments for qgrid settings

_set_grid_options(***kwargs*)
qgrid grid_options kwargs apply to values

Parameters **kwargs** – option values are updated by kwargs

usage:

```
grid_options(editable=True)
```

view(*df_slice*, *handler=True*, *minor=None*, ***kwargs*)

display dataframe with qgrid qgrid display settings defined in inherited Base class

Parameters

- **df_slice** – Dataframe to be displayed
- **handler** – bool, default True, turns on/off preview below grid
- **minor** – list, columns to display, default None

Keyword Arguments

- **show_toolbar** – bool, default True
- **sortable** – bool, default True, causes glitch with editable
- **filterable** – bool, default True, causes glitch with editable
- **forceFitColumns** – bool, default True
- **rowHeight** – int, default 70

- **defaultColumnWidth** – int, default 150
- **maxVisibleRows** – int, default 15
- **minVisibleRows** – int, default 8
- **height** – str, default "250px"

Returns Jupyter Notebook display with a header, qgrid, footer and selected record block

usage:

```
from ozcore import core
core.grid.view(core.dummy.emp, rowHeight=30, handler=False)
```

CHAPTER

NINE

AG-GRID EXTENDED

ipyaggrid is designed to enable easy access to basic ag-Grid features but allows customization through a full access to the ag-Grid API.

Hint: This grid is very useful for data mangling with Pandas.

9.1 Basic aggrid usage:

```
from ozcore import core
core.gridag.view(df)
```

9.2 Available themes:

- ag-theme-balham
- ag-theme-balham-dark
- ag-theme-material
- ag-theme-fresh
- ag-theme-dark
- ag-theme-blue
- ag-theme-bootstrap
- ag-theme-excel

9.3 Module aggrid

AG Grid python wrapper

class ozcore.core.aggrid.aggrid.**Grid**
wrapper class for ipyaggrid

AG Grid integration with Jupyter Notebook

__init__()
parameters for grid options

view(*df*, ***kwargs*)
view a dataframe with AG Grid

Parameters **df** – Dataframe

Keyword Arguments

- **groupby** – list|str, define fields to group by
- **enableSorting** – bool, default True
- **enableFilter** – bool, default True
- **enableColResize** – bool, default True
- **enableRangeSelection** – bool, default True
- **enableRangeHandle** – bool, default True
- **quick_filter** – bool, default True
- **show_toggle_edit** – bool, default False
- **export_mode** – str, default "disabled"
- **export_csv** – bool, default False
- **export_excel** – bool, default False
- **show_toggle_delete** – bool, default False
- **keep_multiindex** – bool, default False
- **index** – bool, default False
- **theme** – str, default 'ag-theme-balham'
- **columns_fit** – str, default 'auto'

Returns a dataframe displayed with AG Grid in a Jupyter Notebook

CHAPTER

TEN

MS DOCX FILE HELPERS

class ozcore.core.office.docx.docx.**Docx**

Microsoft docx files helper class

usage:

```
from ozcore.core import office
office.word.combine_docx_files(folder="some-folder")
```

combine_docx_files(*folder: Optional[Union[str, pathlib.PosixPath]] = None*, *files: Optional[Union[List[str], List[pathlib.PosixPath]]] = None*, *save_to_folder: Optional[Union[str, pathlib.PosixPath]] = None*)

Combine multiple documents (.docx, .doc, .docxm)

Parameters

- **folder** – str|PosixPath, default None
- **files** – list(str)|list(PosixPath), default None
- **save_to_folder** – str|PosixPath, default None

> **Warning:** You can either define `folder``or ``files`. If `files` given (more than one), `save_to_folder` must be defined.

Returns

- combines docx files and saves in `save_to_folder`
- if `folder` is given but no `save_to_folder` , output file saved in `folder`

CHAPTER

ELEVEN

DEVELOPMENT ENVIRONMENT

Todo: dev environment

CHAPTER

TWELVE

OZCORE

OzCore is my core.

It is automating my boring stuff. A time saver for me. I can access frequently used modules and methods easliy. Most of my time is passing with Jupyter Notebooks, and OzCore is my best friend.

Many code snippets derive from a hard processes. I search for the best fitting options and try them sometimes for hours or days. OzCore keeps my good practices as side notes. My quality time for coding is mostly passing with annoying dev-environment, re-setups and glitches. OzCore skips the hard process and provides me with a fresh working environment, where All necessary packages installed.

12.1 Goals

- a Jupyter Notebook having the most used modules.
- **shorthand to**
  - path operations
  - tmp folder actions
  - Sqlite operations
  - CSV operations
  - Dataframe operations
  - Dummy records
  - Jupyter initial setups
  - Jupyter Notebook grid plugins
  - and some MS Office automations

12.2 Warnings

12.2.1 Work In Progress

This package is continuously WIP. It is for my development projects and I happily share it with open source developers. But, please bear with the versions and tests, which may effect your applications.

12.2.2 Massive Dependencies

Since OzCore is a collection of snippets using diverse packages, massive amount of dependencies will be downloaded.

> **Warning:** Please see dependencies in `pyproject.toml` before installing.

12.2.3 MacOS bound modules

Some of the helper modules and functions are directly referenced to MacOS environment. Espacially Windows users may not like it. And some references are pointing to options which may not be available in your system. Such as OneDrive folder or gDrive folder. I have tests to distinguish between users, nevertheless you should be aware of this.

12.3 Installation

I would prefer to run on an Anaconda environment. Here you will find multiple examples.

> **Warning:** Python environment management has become a disaster. Please be sure where you are with `which python`.

12.3.1 I. Anaconda

You may want to set the global Python version with Pyenv as `pyenv global 3.8.3` (of course if pyenv is available.)

```
# create new env
conda create -n py383

conda activate py383

# this initiates every source bindings to new env
conda install pip

# install ozcore
conda install ozcore
```

12.3.2 II. Virtualenv

```
# create a virtualenv
python -m venv .venv

source .venv/bin/activate

pip install ozcore
```

12.3.3 III. Pip simple

```
# in any environment having pip
pip install ozcore
```

12.3.4 IV. Poetry with Pyenv

```
# in any package folder (3.8.4 version of python is arbitrary)
pyenv local 3.8.4

poetry shell

poetry add ozcore
```

12.3.5 V. GitHub with Pip

```
# in any environment having pip
pip install https://github.com/ozgurkalan/OzCore.git
```

12.3.6 VI. GitHub clone

```
# in some folder, e.g. Desktop
git clone https://github.com/ozgurkalan/OzCore.git
```

12.4 Jupyter Kernel

Jupyter has its own configuration. Espacially when you have Anaconda installed, `kernel.json` may have what conda sets.

For your Jupyter Notebook to run in your dedicated environment, please use the following script:

```
# add kernell to Jupyter
python -m ipykernel install --user --name=<your_env_name>

# remove the kernel from Jupyter
jupyter kernelspec uninstall <your_env_name>
```

Fresh installs may have problems with enabling extentions. You shall run the commands below to activate.

```
jupyter nbextension enable --py --sys-prefix qgrid
jupyter nbextension enable --py --sys-prefix widgetsnbextension
```

12.5 Jupyter Extensions

This step copies the `nbextensions` javascript and css files into the jupyter server's search directory, and edits some jupyter config files.

```
jupyter contrib nbextension install --user
```

12.6 CHANGELOG

12.6.1 v1.1.11

- Include unzip helper function
- Add dependencies: pytest_httpserver, typer, tqdm
- config and Makefile for pre-publish actions of the package
- ozcore.__version__ is available

12.6.2 v1.1.10

Include seaborn in dependencies which pulled scipy as dependecy. Corrected installation on a conda environment in README.

12.6.3 v1.1.9

Include jupyter_contrib_nbextensions in dependencies. Update some of the docstrings. Include handcrafted requirements.txt in docs folder to meet ReadTheDocs build requirements.

12.6.4 v1.1.8

Reduce number of modules. Fit to ReadTheDocs requirements and include a requirements.txt derived from poetry.

12.6.5 v1.1.7

Rearrange and fix docs

12.6.6 v1.1.6

A stable version after experimenting PyPi automation with Github actions.

12.6.7 v1.0.1

Initial commit

PYTHON MODULE INDEX

O

INDEX

Symbols

A

B

C

D

E

F

G

I

J

L

M

N

O

P

R

S

T

U

V