korexo_profile documentation

Release 0.5.post11+g635a229

DEW Water Science (Kent Inverarity)

May 29, 2023

CONTENTS:

Python code to load sonde profile CSV files produced by the KorEXO sondes.

Source code is hosted on GitHub.

CHAPTER

ONE

HOW TO USE

Example file:

```
KorEXO MEASUREMENT DATA FILE EXPORT,,,,,,,,,,,,,,,,,,,,,
,,,,,,,,,,,,,,,,,,,,,
FILE CREATED:,10/16/2020 6:21:53 AM,,,,,,,,,,,,,,,,,,,,
,,,,,,,,,,,,,,,,,,,,,
,,,MEAN VALUE:,1172.5,0.605,1277.2,85.5,85.6,7.66,47.1,0.859,0.64,1268.1,824,NA,-126,20.
↪679,0.812,3.13,11.4,
,,,STANDARD DEVIATION:,566.7,0.791,617.1,9.2,9.2,0.97,1.1,1.123,0.31,612.7,398,0.1,5.7,0.
↪8,0.972,0,0,
,,,,,,,,,,,,,,,,,,,,,
,,,SENSOR SERIAL NUMBER:,19A103955,19B104242,19A103955,19D101830,19D101830,19D101830,
↪19B105042,19B104242,19A103955,19A103955,19A103955,19B105042,19B105042,19A103955,
↪19B104242,19C000969,19C000969,
Date (MM/DD/YYYY),Time (HH:mm:ss),Time (Fract. Sec),Site Name,Cond µS/cm,Depth m,nLF␣
↪Cond µS/cm,ODO % sat,ODO % local,ODO mg/L,ORP mV,Pressure psi a,Sal psu,SpCond µS/cm,
↪TDS mg/L,pH,pH mV,Temp °C,Vertical Position m,Battery V,Cable Pwr V,DTW
10/16/2020,15:49:33,0,SQR097,3,-0.001,3.4,102.3,102.5,9.5,49.6,-0.002,0,3.4,2,9.23,-137.
↪6,18.996,-0.002,3.13,11.4,4.2
10/16/2020,15:49:34,0,SQR097,3,-0.001,3.4,102.3,102.5,9.5,49.5,-0.002,0,3.4,2,9.23,-137.
↪6,18.998,-0.001,3.13,11.4,
10/16/2020,15:49:35,0,SQR097,3,-0.001,3.4,102.3,102.5,9.49,49.4,-0.002,0,3.4,2,9.23,-137.
↪6,18.999,-0.001,3.13,11.4,
10/16/2020,15:49:36,0,SQR097,3,-0.001,3.4,102.3,102.5,9.49,49.3,-0.002,0,3.4,2,9.23,-137.
↪6,19.001,-0.001,3.13,11.4,
10/16/2020,15:49:37,0,SQR097,3,-0.001,3.4,102.3,102.5,9.5,49.2,-0.002,0,3.4,2,9.23,-137.
↪6,19.002,-0.001,3.13,11.4,
10/16/2020,15:49:38,0,SQR097,3,-0.001,3.4,102.3,102.5,9.49,49.1,-0.002,0,3.4,2,9.23,-137.
↪6,19.004,-0.001,3.13,11.4,
10/16/2020,15:49:39,0,SQR097,3,-0.001,3.4,102.4,102.5,9.49,48.9,-0.002,0,3.3,2,9.23,-137.
↪6,19.005,-0.001,3.13,11.4,
10/16/2020,15:49:40,0,SQR097,3,-0.001,3.4,102.4,102.5,9.5,48.8,-0.002,0,3.3,2,9.23,-137.
↪6,19.007,-0.002,3.13,11.4,
10/16/2020,15:49:41,0,SQR097,3,-0.001,3.4,102.4,102.5,9.49,48.7,-0.002,0,3.3,2,9.23,-137.
↪6,19.008,-0.002,3.13,11.4,
```

To read this file:

[1]:
```
import korexo_profile
data = korexo_profile.read("../tests/example2.csv")
```

This returns a dictionary with three keys:

[2]:
```
data.keys()
```

[2]:
```
dict_keys(['metadata', 'datasets', 'dataframe'])
```

Sometimes, people open the raw file in Excel to view or edit it in some small way and then save as a CSV. This seems innocuous, but completely changes the format (and usually the encoding also - see below).

1.1 Raw pandas dataframe

The raw data is read into a pandas dataframe:

[3]:
```
df = data["dataframe"]
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9 entries, 0 to 8
Data columns (total 22 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   Date (MM/DD/YYYY)    9 non-null      object
 1   Time (HH:mm:ss)      9 non-null      object
 2   Time (Fract. Sec)    9 non-null      int64
 3   Site Name            9 non-null      object
 4   Cond µS/cm           9 non-null      int64
 5   Depth m              9 non-null      float64
 6   nLF Cond µS/cm       9 non-null      float64
 7   ODO % sat            9 non-null      float64
 8   ODO % local          9 non-null      float64
 9   ODO mg/L             9 non-null      float64
 10  ORP mV               9 non-null      float64
 11  Pressure psi a       9 non-null      float64
 12  Sal psu              9 non-null      int64
 13  SpCond µS/cm         9 non-null      float64
 14  TDS mg/L             9 non-null      int64
 15  pH                   9 non-null      float64
 16  pH mV                9 non-null      float64
 17  Temp °C              9 non-null      float64
 18  Vertical Position m  9 non-null      float64
 19  Battery V            9 non-null      float64
 20  Cable Pwr V          9 non-null      float64
 21  DTW                  1 non-null      float64
dtypes: float64(15), int64(4), object(3)
memory usage: 1.7+ KB
```

[4]:
```
df.head()
```

[4]:
```
  Date (MM/DD/YYYY) Time (HH:mm:ss)  Time (Fract. Sec) Site Name  Cond µS/cm  \
0        10/16/2020        15:49:33                 0    SQR097           3
1        10/16/2020        15:49:34                 0    SQR097           3
2        10/16/2020        15:49:35                 0    SQR097           3
3        10/16/2020        15:49:36                 0    SQR097           3
```

(continues on next page)

(continued from previous page)

```
4        10/16/2020        15:49:37                  0    SQR097           3

   Depth m  nLF Cond µS/cm  ODO % sat  ODO % local  ODO mg/L  ...  Sal psu  \
0   -0.001             3.4      102.3        102.5      9.50  ...        0
1   -0.001             3.4      102.3        102.5      9.50  ...        0
2   -0.001             3.4      102.3        102.5      9.49  ...        0
3   -0.001             3.4      102.3        102.5      9.49  ...        0
4   -0.001             3.4      102.3        102.5      9.50  ...        0

   SpCond µS/cm  TDS mg/L    pH  pH mV  Temp °C  Vertical Position m  \
0           3.4         2  9.23 -137.6   18.996               -0.002
1           3.4         2  9.23 -137.6   18.998               -0.001
2           3.4         2  9.23 -137.6   18.999               -0.001
3           3.4         2  9.23 -137.6   19.001               -0.001
4           3.4         2  9.23 -137.6   19.002               -0.001

   Battery V  Cable Pwr V  DTW
0       3.13         11.4  4.2
1       3.13         11.4  NaN
2       3.13         11.4  NaN
3       3.13         11.4  NaN
4       3.13         11.4  NaN

[5 rows x 22 columns]
```

1.2 Per dataset (recommended)

The information is also stored per-dataset, with some data-type conversions applied under the `"datasets"` key. I would recommend using this in most cases. Each item is a dictionary with keys as shown below.

The Date, Time, and Site Name columns have the handy `"median"` key, while all others you'd be more interested in the `"data"` key which contains a numpy `ndarray` with the data:

[5]:
```
data["datasets"][0]
```

[5]:
```
{'name': 'Date',
 'column': 'Date (MM/DD/YYYY)',
 'sensor': '',
 'mean': <NA>,
 'stdev': <NA>,
 'data': [datetime.date(2020, 10, 16),
  datetime.date(2020, 10, 16),
  datetime.date(2020, 10, 16),
  datetime.date(2020, 10, 16),
  datetime.date(2020, 10, 16),
  datetime.date(2020, 10, 16),
  datetime.date(2020, 10, 16),
  datetime.date(2020, 10, 16),
  datetime.date(2020, 10, 16)],
 'median': datetime.date(2020, 10, 16)}
```

[6]:
```
data["datasets"][1]
```

[6]:
```
{'name': 'Time',
 'column': 'Time (HH:mm:ss)',
 'sensor': '',
 'mean': <NA>,
 'stdev': <NA>,
 'data': array(['15:49:33', '15:49:34', '15:49:35', '15:49:36', '15:49:37',
        '15:49:38', '15:49:39', '15:49:40', '15:49:41'], dtype=object),
 'median': '15:49:33'}
```

And the last one:

[7]:
```
data["datasets"][9]
```

[7]:
```
{'name': 'ODO mg/L',
 'column': 'ODO mg/L',
 'sensor': '19D101830',
 'mean': 7.66,
 'stdev': 0.97,
 'data': array([9.5 , 9.5 , 9.49, 9.49, 9.5 , 9.49, 9.49, 9.5 , 9.49]),
 'median': 9.49}
```

1.3 Other metadata

Raw file metadata is also available:

[8]:
```
data["metadata"]
```

[8]:
```
{'created_file': datetime.datetime(2022, 4, 7, 16, 45, 10, 874033),
 'modified_file': datetime.datetime(2022, 4, 7, 16, 45, 10, 875029),
 'created_info': '10/16/2020 6:21:53 AM,,,,,,,,,,,,,,,,,,,,',
 'header_line_no': 9,
 'params': ['Cond µS/cm',
  'Depth m',
  'nLF Cond µS/cm',
  'ODO % sat',
  'ODO % local',
  'ODO mg/L',
  'ORP mV',
  'Pressure psi a',
  'Sal psu',
  'SpCond µS/cm',
  'TDS mg/L',
  'pH',
  'pH mV',
  'Temp °C',
  'Vertical Position m',
  'Battery V',
  'Cable Pwr V',
  'DTW'],
 'sensors': ['19A103955',
```

(continues on next page)

(continued from previous page)

```
 '19B104242',
 '19A103955',
 '19D101830',
 '19D101830',
 '19D101830',
 '19B105042',
 '19B104242',
 '19A103955',
 '19A103955',
 '19A103955',
 '19B105042',
 '19B105042',
 '19A103955',
 '19B104242',
 '19C000969',
 '19C000969',
 ''],
'means': [1172.5,
 0.605,
 1277.2,
 85.5,
 85.6,
 7.66,
 47.1,
 0.859,
 0.64,
 1268.1,
 824,
 <NA>,
 -126,
 20.679,
 0.812,
 3.13,
 11.4,
 nan],
'stdevs': [566.7,
 0.791,
 617.1,
 9.2,
 9.2,
 0.97,
 1.1,
 1.123,
 0.31,
 612.7,
 398,
 0.1,
 5.7,
 0.8,
 0.972,
 0,
 0,
```

(continues on next page)

(continued from previous page)

```
 nan]}
```

1.4 Date formats

I have found that the way the KorEXO sondes encode dates to be a little messy. Note that the column header for the first column seems to encode the date format: `Date (MM/DD/YYYY)`. This is detected and used automatically by default. However, I've seen some files produced where the date format is backwards, so you can also choose to specify the appropriate format in Python datetime format specifiers:

For this example:

```
sep=,
KorEXO MEASUREMENT DATA FILE EXPORT

FILE CREATED:,10/12/2019 10:12:33 PM

,,,MEAN VALUE:,601.0,0.298,662.6,86.1,85.6,8.17,-35.2,0.424,0.33,657.3,427,NA,-83.2,18.
↪269,0.348,3.06,11.2
,,,STANDARD DEVIATION:,644.7,0.495,709.6,18.5,18.4,1.95,12.0,0.703,0.35,704.0,458,0.20,
↪11.3,2.257,0.489,0.00,0.0

,,,SENSOR SERIAL NUMBER:,18K105940,19A101644,18K105940,18L102250,18L102250,18L102250,
↪18L101854,19A101644,18K105940,18K105940,18K105940,18L101854,18L101854,18K105940,
↪19A101644,19B100840,19B100840
Date (MM/DD/YYYY),Time (HH:mm:ss),Time (Fract. Sec),Site Name,Cond µS/cm,Depth m,nLF␣
↪Cond µS/cm,ODO % sat,ODO % local,ODO mg/L,ORP mV,Pressure psi a,Sal psu,SpCond µS/cm,
↪TDS mg/L,pH,pH mV,Temp °C,Vertical Position m,Battery V,Cable Pwr V
11/12/2019,08:40:30,0.0,SQR100,9.1,0.002,11.5,98.0,97.4,10.02,11.8,0.002,0.00,11.4,7,7.
↪59,-44.0,14.354,0.003,3.06,11.1
11/12/2019,08:40:31,0.0,SQR100,9.0,0.002,11.5,98.0,97.5,10.02,12.2,0.002,0.00,11.4,7,7.
↪59,-43.8,14.352,0.004,3.06,11.1
11/12/2019,08:40:32,0.0,SQR100,9.0,0.002,11.5,98.1,97.6,10.03,11.0,0.002,0.00,11.3,7,7.
↪59,-43.7,14.349,0.000,3.06,11.1
11/12/2019,08:40:33,0.0,SQR100,9.0,0.002,11.5,98.6,98.1,10.09,8.7,0.003,0.00,11.3,7,7.61,
↪-45.4,14.357,0.003,3.06,11.1
11/12/2019,08:40:34,0.0,SQR100,9.1,0.002,11.5,99.3,98.8,10.16,-9.7,0.003,0.00,11.4,7,7.
↪87,-59.5,14.420,0.001,3.06,11.1
11/12/2019,08:40:35,0.0,SQR100,9.0,0.002,11.5,99.7,99.1,10.18,-14.5,0.003,0.00,11.3,7,7.
↪98,-65.6,14.523,0.001,3.06,11.1
11/12/2019,08:40:36,0.0,SQR100,9.0,0.002,11.4,101.1,100.5,10.30,-21.2,0.002,0.00,11.2,7,
↪8.09,-71.9,14.754,-0.004,3.06,11.1
11/12/2019,08:40:37,0.0,SQR100,9.1,0.002,11.3,102.3,101.8,10.37,-23.1,0.003,0.00,11.2,7,
↪8.13,-73.9,15.171,0.001,3.06,11.1
11/12/2019,08:40:38,0.0,SQR100,9.2,0.002,11.3,102.6,102.1,10.31,-25.5,0.003,0.00,11.2,7,
↪8.16,-75.9,15.666,0.001,3.06,11.1
11/12/2019,08:40:39,0.0,SQR100,9.2,0.002,11.3,102.3,101.7,10.17,-27.3,0.003,0.00,11.1,7,
↪8.20,-78.1,16.020,0.000,3.06,11.2
11/12/2019,08:40:40,0.0,SQR100,9.3,0.002,11.3,101.4,100.8,10.00,-28.3,0.003,0.00,11.1,7,
↪8.22,-79.7,16.220,0.002,3.06,11.2
11/12/2019,08:40:41,0.0,SQR100,9.2,0.002,11.2,100.4,99.9,9.87,-28.5,0.003,0.00,11.1,7,8.
↪24,-80.7,16.250,-0.001,3.06,11.2
```

The default automatic detection:

[9]:
```
data = korexo_profile.read("../tests/example1.csv", datefmt="auto")
data["datasets"][0]["median"]
```

[9]:
```
datetime.date(2019, 11, 12)
```

Specifying it directly should provide the same result:

[10]:
```
data = korexo_profile.read("../tests/example1.csv", datefmt="%m/%d/%Y")
data["datasets"][0]["median"]
```

[10]:
```
datetime.date(2019, 11, 12)
```

But in this case the true date is actually December 11th, which we can force:

[11]:
```
data = korexo_profile.read("../tests/example1.csv", datefmt="%d/%m/%Y")
data["datasets"][0]["median"]
```

[11]:
```
datetime.date(2019, 12, 11)
```

1.5 Beware encodings

I believe the sondes use UTF-16-LE by default. Most text editors do not use this, so if someone has opened the file and then saved it, it likely will have converted it to UTF-8 or Windows-1252.

The default behaviour is simplest:

[12]:
```
data = korexo_profile.read("../tests/example2.csv")
```

This is equivalent internally to:

[13]:
```
data = korexo_profile.read("../tests/example2.csv", encoding="utf-16", auto_revert_
↪encoding="cp1252")
```

Because `auto_revert_encoding` is not False, the code will first check whether example2.csv is UTF-16. If it is not, then the file will be loaded using the value of `auto_revert_encoding` (`'cp1252'`). If it is UTF-16, it will be loaded as such.

If you want the code to simply use the encoding you direct it to, set `auto_revert_encoding=False`. This may result in messy errors and weird behaviour.

1.6 Skip all that… just give me a complete spreadsheet

[14]:
```
data = korexo_profile.read("../tests/example1_full.csv")
df = korexo_profile.convert_datasets_to_df(data["datasets"])
df.head()
```

[14]:

	datetime	date	time	time_sec	site	cond	\
0	2019-11-12 08:40:30	2019-11-12	08:40:30	0.0	SQR100	9.1	
1	2019-11-12 08:40:31	2019-11-12	08:40:31	0.0	SQR100	9.0	
2	2019-11-12 08:40:32	2019-11-12	08:40:32	0.0	SQR100	9.0	
3	2019-11-12 08:40:33	2019-11-12	08:40:33	0.0	SQR100	9.0	

(continues on next page)

(continued from previous page)

```
4 2019-11-12 08:40:34  2019-11-12  08:40:34       0.0  SQR100   9.1

   water_depth  cond_nlf  do_sat  do_local  ...  press  sal_psu  spcond  tds  \
0        0.002      11.5    98.0      97.4  ...  0.002      0.0    11.4    7
1        0.002      11.5    98.0      97.5  ...  0.002      0.0    11.4    7
2        0.002      11.5    98.1      97.6  ...  0.002      0.0    11.3    7
3        0.002      11.5    98.6      98.1  ...  0.003      0.0    11.3    7
4        0.002      11.5    99.3      98.8  ...  0.003      0.0    11.4    7

     ph  ph_mv    temp  vert_pos  battery  cable_power
0  7.59  -44.0  14.354     0.003     3.06         11.1
1  7.59  -43.8  14.352     0.004     3.06         11.1
2  7.59  -43.7  14.349     0.000     3.06         11.1
3  7.61  -45.4  14.357     0.003     3.06         11.1
4  7.87  -59.5  14.420     0.001     3.06         11.1

[5 rows x 22 columns]
```

And let's say the depth to water was 5.15 m, so let's add that offset:

[15]:
```
df["dtw"] = df.water_depth + 5.15
df.head()
```

[15]:
```
             datetime        date      time  time_sec    site  cond  \
0 2019-11-12 08:40:30  2019-11-12  08:40:30       0.0  SQR100   9.1
1 2019-11-12 08:40:31  2019-11-12  08:40:31       0.0  SQR100   9.0
2 2019-11-12 08:40:32  2019-11-12  08:40:32       0.0  SQR100   9.0
3 2019-11-12 08:40:33  2019-11-12  08:40:33       0.0  SQR100   9.0
4 2019-11-12 08:40:34  2019-11-12  08:40:34       0.0  SQR100   9.1

   water_depth  cond_nlf  do_sat  do_local  ...  sal_psu  spcond  tds    ph  \
0        0.002      11.5    98.0      97.4  ...      0.0    11.4    7  7.59
1        0.002      11.5    98.0      97.5  ...      0.0    11.4    7  7.59
2        0.002      11.5    98.1      97.6  ...      0.0    11.3    7  7.59
3        0.002      11.5    98.6      98.1  ...      0.0    11.3    7  7.61
4        0.002      11.5    99.3      98.8  ...      0.0    11.4    7  7.87

   ph_mv    temp  vert_pos  battery  cable_power    dtw
0  -44.0  14.354     0.003     3.06         11.1  5.152
1  -43.8  14.352     0.004     3.06         11.1  5.152
2  -43.7  14.349     0.000     3.06         11.1  5.152
3  -45.4  14.357     0.003     3.06         11.1  5.152
4  -59.5  14.420     0.001     3.06         11.1  5.152

[5 rows x 23 columns]
```

Then re-index so the profile is at 5 cm intervals:

[16]:
```
df2 = korexo_profile.make_regularly_spaced(df, "dtw", step=0.05)
df2.head()
```

[16]:
```
          time_sec         cond  water_depth     cond_nlf     do_sat  \
dtw
```

(continues on next page)

(continued from previous page)

```
5.150000       0.0     9.533333     0.000000    11.500000  91.700000
5.202174       0.0  1203.097516     0.052174  1324.406211  92.240373
5.254348       0.0  1220.579710     0.104348  1343.950483  91.725845
5.306522       0.0  1225.072347     0.156522  1348.894872  91.595905
5.358696       0.0  1225.987800     0.208696  1349.886039  91.582138

           do_local   do_conc     orp_mv     press   sal_psu       spcond  \
dtw
5.150000  91.200000  8.910000 -39.200000  0.001000  0.000000    11.366667
5.202174  91.670186  8.264037 -42.380745  0.073963  0.660000  1313.976398
5.254348  91.225845  8.212585 -41.203382  0.148539  0.670000  1333.350483
5.306522  91.095905  8.199795 -40.589761  0.222054  0.670205  1338.192824
5.358696  91.082138  8.199107 -40.555346  0.296185  0.670893  1339.177108

                tds        ph      ph_mv       temp  vert_pos  battery  \
dtw
5.150000    7.000000  8.390000 -89.133333  16.663667  0.001333     3.06
5.202174  854.192547  8.364037 -88.661491  20.583807  0.419938     3.06
5.254348  866.483092  8.305169 -85.658454  20.573034  0.694548     3.06
5.306522  870.184295  8.289795 -84.583618  20.575123  0.807270     3.06
5.358696  870.803774  8.289107 -84.528553  20.575536  0.811537     3.06

          cable_power
dtw
5.150000         11.2
5.202174         11.2
5.254348         11.2
5.306522         11.2
5.358696         11.2
```

CHAPTER

TWO

API DOCUMENTATION

korexo_profile.**read**(*fn*, *encoding='utf-16'*, *parse_dts=True*, *datefmt='auto'*, *auto_revert_encoding='cp1252'*)

Read KorEXO sonde profile.

Parameters

- **fn** (*str*) – filename
- **encoding** (*str*) – file encoding. I believe raw KorEXO output is in UTF-16.
- **parse_dts** (*bool*) – whether to attempt to parse datetimes or not.
- **datefmt** (*str*) – use "auto" to use the column header to infer the date format, although note that this isn't always correct. In that case, set it here using Python datetime string formats e.g. `"%d/%m/%Y"`
- **auto_revert_encoding** (*str or None/False*) – attempt to check whether the file is UTF-16 and if it is not i.e. there is no BOM, then use whatever encoding is set here. Set to `False` only if you want the code to attempt *encoding* and fail messily if it is not.

Returns

with keys 'metadata', 'dataframe' and 'datasets'. See package documentation for more details.

Return type

dict

korexo_profile.**convert_datasets_to_df**(*datasets*, *mapping={'Battery V': 'battery', 'Cable Pwr V': 'cable_power', 'Cond µS/cm': 'cond', 'Date (DD/MM/YYYY)': 'date', 'Date (MM/DD/YYYY)': 'date', 'Depth m': 'water_depth', 'ODO % local': 'do_local', 'ODO % sat': 'do_sat', 'ODO mg/L': 'do_conc', 'ORP mV': 'orp_mv', 'Pressure psi a': 'press', 'Sal psu': 'sal_psu', 'Site Name': 'site', 'SpCond µS/cm': 'spcond', 'TDS mg/L': 'tds', 'Temp °C': 'temp', 'Time (Fract. Sec)': 'time_sec', 'Time (HH:mm:ss)': 'time', 'Vertical Position m': 'vert_pos', 'nLF Cond µS/cm': 'cond_nlf', 'pH': 'ph', 'pH mV': 'ph_mv'}*)

Convert a list of datasets to a dataframe, include renaming of column names if desired.

Parameters

- **datasets** (*list*) – see output of *korexo_profile.read()*.
- **mapping** (*dict*) – optional. The default mapping is stored in korexo_profile.COL_MAPPING

Returns

dataframe with "datetime" column added.

Return type
pandas.DataFrame

korexo_profile.**make_regularly_spaced**(*df*, *index_col='dtw'*, *step=0.05*, *step_precision=5*)

Convert dataframe to regular spacing based on an index column.

Parameters

- **df** (*pandas DataFrame*) –
- **index_col** (*str*) – column of *df* for which a regularly-spaced set of values should be created at *step* and then all other data interpolated against.
- **step** (*float*) – interval desired in *index_col*
- **step_precision** (*int*) –

Returns
dataframe, where the newly created *index_col* values are set as the dataframe index. All other columns of *df* are included as columns of the *df*, interpolated at the new *index_col* values.

Return type
pandas.DataFrame

korexo_profile.**to_las**(*df*, *fn*, *encoding='utf-16'*, *col_metadata=None*, *well_metadata=None*, *param_metadata=None*, *add_mtime_date='DATE'*, *auto_revert_encoding='cp1252'*)

Convert a KorEXO profile file to Log ASCII Standard (LAS).

Parameters

- **df** (*pandas DataFrame*) – a regularly-spaced output from reading a KorEXO profile CSV file.
- **fn** (*str*) – the original CSV file
- **col_metadata** (*dict*) – optional metadata for the columns. The keys should refer to columns of *df* and each value should be a tuple. The first item of the tuple is a string for the unit e.g. `"m"`, and the second item is the description.
- **well_metadata** (*dict*) – dict of metadata to add to the LAS file's ~Well section.
- **param_metadata** (*dict*) – dict of metadata to add to the LAS file's ~Param section
- **add_mtime_date** (*str*) – add the file modified time of *fn* as a value in the ~Well section. Set to False or None to prevent adding it at all.
- **auto_revert_encoding** (*bool*) – attempt to check whether the file is UTF-16 and if it is not i.e. there is no BOM, then use this encoding instead. Set to `False` only if you want the code to fail messily if you have the encoding wrong.

Returns
lasio.LASFile object

The contents of the original KorEXO profile CSV file will be recorded in the LAS file's ~Other block.

Example:

```
>>> import korexo_profile
>>> data = korexo_profile.read(fn, datefmt="%d/%m/%Y")
>>> df = korexo_profile.convert_datasets_to_df(data["datasets"])
>>> df["water_depth"] += WELL_DEPTH_TO_WATER_MEASUREMENT
>>> df2 = korexo_profile.make_regularly_spaced(df, "water_depth", step=0.05)
>>> las = korexo_profile.to_las(df2, fn)
```

CHAPTER

THREE

DEVELOPMENT

An issue tracker is located at GitLab.

The documentation is accessible on GitLab when you are signed in.

Contact Kent Inverarity for access to the GitLab site.

3.1 To build and publish a release

If necessary, create a new version with a git tag per setuptools-scm:

```
$ git log --oneline
0ad38e0 (HEAD -> master) update everything
13a8644 (tag: v0.2) update
ab86ff2 (tag: v0.1) Initial commit
$ git tag v0.3
```

Then the usual way to build a wheel:

```
$ python setup.py bdist_wheel
```

And to publish, the usual:

```
$ twine upload dist\korexo_profile-0.3-py3-none-any.whl
```

INDEX