akiFlagger

Release 0.0.3

Jan 25, 2022

Contents


AKI FlaggeR

CHAPTER 1

Introduction

Acute Kidney Injury (AKI) is a sudden onset of kidney failure and damage marked by an increase in the serum creatinine levels (amongst other biomarkers) of the patient. Kidney Disease Improving Global Outcomes (KDIGO) has a set of guidelines and standard definitions of AKI:

- *Stage 1:* 50% increase in creatinine in < 7 days or 0.3 increase in creatinine in < 48 hours
- *Stage 2:* 100% increase in (or doubling of) creatinine in < 48 hours
- *Stage 3:* 200% increase in (or tripling of) creatinine in < 48 hours

This package contains a flagger to determine if a patient has developed AKI based on longitudinal data of serum creatinine measurements. More information about the specific data input format can be found in the *Getting started* section.

1.1 Methods of calculating AKI

There are two methods to retroactively determine if a patient developed AKI: `rolling-window` and `back-calculation`.

`Rolling` `Window (default)`
: The rolling window definition of AKI is based on the change in creatinine in a 48 hour or 7 day rolling window period. These are the stages mentioned in the KDIGO guidelines in the *Introduction* above.

`Historical` `baseline trumping (back-calculate)`
: The idea with Historical baseline trumping is to use the historical baseline creatinine value as the value to compare the current creatinine to when runnning the KDIGO criterion *instead of* the rolling window value; i.e. the historical baseline *trumps* the rolling minimum value.

  Definition: The historical baseline is defined as the median of the patient's outpatient creatinine values from 365 to 7 days prior to admission.

  Reasoning: Right when a patient is admitted to the hospital, their creatinine might not be representative of what their true, stable creatinine values normally are. As such, you might want to use the *historical baseline value*. This historical baseline value, calculated retroactively, is only used around the time of admission - specifically

from the time of admission to 7 days (+ padding) out. For any time outside of this, the rolling window is still in effect... but this allows you to capture patients whose hemodynamic balance might be messed up at the time of admission.

If there are no outpatient creatinine values measured for the patient from 365 t o 7 days prior to admission, it is possible to still impute a baseline creatinine value based on the patients demographics: namely their age, sex, and race. This is what the `eGFR_impute` option in the flagger does: *eGFR imputation* because it assumes an eGFR of 75 mL/min/1.73m^2 and estimates the creatinine from that.

CHAPTER 2

Installation

Python

You can install the flagger with `pip`. Simply type the following into command line and the package should install properly.

```
pip install akiFlagger
```

To ensure that it is working properly, you can open a Python session and test it with.

```
import akiFlagger

akiFlagger.__version__

>> '1.0.0'
```

R

You can install the flagger through CRAN. Simply type the following into an RStudio terminal and the package should install properly.

```
install.packages('akiFlagger')
```

CHAPTER 3

Getting started

This package is meant to handle patient data. Let's walk through an example of how to use this package with some toy data since real patient data is probably protected health information.

Once you've installed the package following the instructions in *Installation*, you're ready to get started. To begin with, we'll import the `akiFlagger` module.

Python

```
import akiFlagger

print(akiFlagger.__version__)

from akiFlagger import AKIFlagger, generate_toy_data

>> '1.0.0'
```

R

```
library(akiFlagger)

?returnAKIpatients

>  Rendering development documentation for 'returnAKIpatients'
```

3.1 Let's start off by creating some toy data.

Python

The flagger comes with a built-in generator of a toy dataset to demonstrate how it works. Simply call the *generate_toy_data()* function. By default, the toy dataset has 100 patients, but let's initialize ours with 1000 patients.

```
toy = generate_toy_data(num_patients=1000)

print('Toy dataset shape: {}'.format(toy.shape))

>> Successfully generated toy data!

   Toy dataset shape: (9094, 6)
```

The toy dataset comes with columns for the patient identifier, the encounter identifier, whether the measurement was inpatient or outpatient, the creatinine measurement and time at which the measurement was taken. `toy.head()` should yield something like this:

	patient_id	inpatient	time	creatinine
0	12732	False	2020-02-23 23:42:42	1.06
1	12732	False	2020-02-24 23:42:42	1.26
2	12732	False	2020-02-27 05:42:42	1.05
3	12732	True	2020-03-01 17:42:42	1.42
4	12732	True	2020-03-03 05:42:42	1.61

R

The R package comes with a built-in dataset, *toy*. The toy dataset comes with columns for the patient identifier, inpatient, the creatinine measurement and the time at which the measurement was taken. `head(toy)` should yield something like this:

	patient_id	inpatient	time	creatinine
1	12732	FALSE	2019-11-16 05:42:42	1.05
2	12732	FALSE	2019-11-20 05:42:42	1.61
3	12732	FALSE	2020-01-15 05:42:42	1.42
4	12732	FALSE	2020-02-27 11:42:42	1.26
5	12732	TRUE	2020-03-01 17:42:42	1.06
6	19845	FALSE	2019-11-20 18:02:54	0.89

Tip!

In order to calculate AKI, the flagger expects a dataset with certain columns in it. Depending on the type of computation you are interested in, your dataset will need to have different columns. Here's a brief rundown of the necessary columns.

- *Rolling-window*: **patient_id**, **inpatient**, **time**, and **creatinine**
- *Back-calculate*: **patient_id**, **inpatient**, **time**, and **creatinine**
- *eGFR-imputed baseline creatinine*: **age**, **sex** (defaults to female), and **race** (defaults to black).

By default, the naming system is as follows:

patient_id → **'patient_id'**

inpatient/outpatient → **'inpatient'**

creatinine →**'creatinine'**

time → **'time'**

If you have different names for your columns, you **must specify them.**

3.2 Example: Rolling-window

The next code block runs the flagger and returns those patients who satisfy the AKI conditions according to the KDIGO guidelines for change in creatinine values by the rolling-window definition, categorized as follows:

Stage 1: **(1)** 50% ↑ in creatinine in < 7 days OR **(2)** 0.3 mg/dL ↑ in creatinine in < 48 hours

Stage 2: 100% ↑ (or doubling of) in creatinine in < 7 days

Stage 3: 200% ↑ (or tripling of) in creatinine in < 7 days

Python

```
flagger = AKIFlagger()

out = flagger.returnAKIpatients(toy)

out = out.reset_index() # By default, the returned output has the patient_id and time
↪as hierarchical indices

out.head()
```

We can take a look at what our dataframe looks like. `out.head()` yields this:

patient_id	time	inpatient	creatinine	aki
12732	2020-02-23 17:42:42	False	1.42	0
12732	2020-02-28 17:42:42	True	1.26	0
12732	2020-02-29 23:42:42	True	1.05	0
12732	2020-03-02 17:42:42	True	1.61	1
19845	2020-05-08 00:02:54	False	0.76	0

Notice that the dataframe looks exactly the same as we inputted into the flagger save an extra column added, *aki*. This column has values of either 0, 1, 2, or 3, depending on which stage AKI the flagger found. The flagger runs on a row-wise basis, meaning that each row is checked for the increase in creatinine. Should, for example, a patient meet the criterion multiple times within a single encounter, the flagger will flag each measurement as a case of AKI.

Warning: The column names specified within the flagger should match the dataset exactly. The full list of acceptable names can be found in the *returnAKIpatients()* function in the genindex section. For certain cases, the flagger understands special names. For example, *sex = 'male'* will autoconvert the sex column from female to male. But you still need to have a column named *male* in your data frame, otherwise an error will occur.

We can take a look at what the flagger flagged as AKI. `out[out.aki > 0].head()` should give a list of some patients which were flagged. From that, we can subset the dataset on any given patient:

```
out[out.aki > 0].head() # this will give the rows which were marked as AKI by the
↪flagger
out[out.patient_id == 19845] # from that, we can find which patients were flagged
↪with AKI
```

	patient_id	time	inpatient	creatinine	aki
4	19845	2020-05-08 00:02:54	False	0.76	0
5	19845	2020-05-08 06:02:54	False	0.89	0
6	19845	2020-05-09 18:02:54	False	1.07	1
7	19845	2020-05-12 18:02:54	True	0.43	0
8	19845	2020-05-13 18:02:54	True	0.34	0
9	19845	2020-05-14 18:02:54	True	1.12	3

Notice how as we would expect, when the creatinine more than tripled from 0.34 to 1.12, the flagger correctly identified it as Stage 3 AKI.

You can even look at aggregate counts if you wanted as follows (but don't take the numbers too seriously, of course, because this is toy data):

```
aki_counts = out.aki.value_counts()
print('AKI counts')
print('----------')
print('No AKI: {}\nStage 1: {}\nStage 2: {}\nStage 3: {}'.format(aki_counts[0], aki_
↪counts[1], aki_counts[2], aki_counts[3]))

>>  AKI counts
    ----------
    No AKI: 571
    Stage 1: 211
    Stage 2: 99
    Stage 3: 70
```

You can play around with the output of the `returnAKIpatients()` function in-depth to get a better understanding of how the flagger is operating. There are even optional parameters such as `add_min_creat = True` within the flagger which includes some of the intermediate steps the flagger is generating along to calculate AKI. Next, we'll take a look at an example of the other AKI-calculation method, the back-calculation method.

R

```
library(akiFlagger)

out <- returnAKIpatients(toy)

head(out)
```

We can take a look at what the flagger returns. `head(out)` should return:

	patient_id	inpatient	creatinine	time	aki
1	19008	FALSE	2.05	2019-07-08 14:37:33	0
2	19008	FALSE	1.65	2019-07-09 08:37:33	0
3	19008	FALSE	1.58	2019-07-29 08:37:33	0
4	19008	FALSE	1.77	2019-08-10 08:37:33	0
5	19008	FALSE	1.47	2019-09-25 02:37:33	0
6	19008	FALSE	1.64	2019-11-25 14:37:33	0

Notice that the dataframe looks exactly the same as we inputted into the flagger save an extra column added, *aki*. This column has values of either 0, 1, 2, or 3, depending on which stage AKI the flagger found. The flagger runs on a row-wise basis, meaning that each row is checked for the increase in creatinine. Should, for example, a patient meet the criterion multiple times within a single encounter, the flagger will flag each measurement as a case of AKI.

Warning: The patient dataset you input should have minimally these columns: `patient_id`, `inpatient`, `time`, and `creatinine`. If you are interested in demographic-based imputation, you'll also want to include the columns `age`, `sex`, and `race`.

We can take a look at what the flagger flagged as AKI. `head(out[out$aki > 0])` should give a list of some patients which were flagged. From that, we can subset the dataset on any given patient:

```
head(out[out$aki > 0])

out[out$patient_id == 13264]
```

	patient_id	inpatient	creatinine	time	aki
1	13264	FALSE	0.47	2019-07-22 23:16:57	0
2	13264	FALSE	0.1	2019-08-06 23:16:57	0
3	13264	FALSE	0.75	2019-08-11 17:16:57	3
4	13264	FALSE	0.79	2019-08-23 11:16:57	0
5	13264	FALSE	0.61	2019-09-02 17:16:57	0
6	13264	FALSE	0.59	2019-09-03 05:16:57	0
7	13264	FALSE	0.55	2019-09-19 05:16:57	0
8	13264	FALSE	0.49	2019-10-04 17:16:57	0
9	13264	FALSE	0.18	2019-10-09 23:16:57	0
10	13264	FALSE	0.27	2019-11-02 17:16:57	0
11	13264	FALSE	0.5	2019-11-07 05:16:57	1
12	13264	FALSE	0.63	2019-11-08 23:16:57	2
13	13264	FALSE	0.29	2019-11-12 05:16:57	0
14	13264	FALSE	0.22	2019-12-15 11:16:57	0
15	13264	TRUE	0.28	2020-01-12 05:16:57	0

Notice how as we would expect, when the creatinine more than tripled from 0.1 to 0.72, the flagger correctly identified it as Stage 3 AKI. Additionally, row 11 was flagged as stage 1 because that was a greater than 50% increase from 0.27 and row 12 was flagged because it was a greater than 100% increase from 0.27. Even though the flagger is performing a row-wise computation, it is comparing the current creatinine value with the minimum in the past `window1` hours (defaults to 48 hours).

You can look at aggregate counts if you wanted as follows (but don't take the numbers too seriously, of course, because this is toy data):

```
table(out$aki)

>>    0    1    2    3
    1001   44   19   14
```

3.3 Example: Back-calculation

Next, we'll run the flagger to "back-calculate" AKI; that is, using the **median outpatient creatinine values from 365 to 7 days prior to admission** to impute a baseline creatinine value. Then, we'll run the same KDIGO criterion (except for the 0.3 increase) comparing the creatinine value to baseline creatinine.

Python

```
flagger = AKIFlagger(HB_trumping = True, add_baseline_creat = True)

out = flagger.returnAKIpatients(toy)

out.head()
```

patient_id	time	inpatient	creatinine	baseline_creat	aki
12732	2020-02-22 23:42:42	False	1.26		0
12732	2020-02-24 05:42:42	False	1.61		1
12732	2020-02-24 23:42:42	False	1.05		0
12732	2020-02-26 23:42:42	False	1.42		1
12732	2020-03-03 11:42:42	True	1.06		0

R

```
out <- returnAKIpatients(toy, HB_trumping = T, add_baseline_creat = T)

head(out)
```

	patient_id	inpatient	creatinine	time	baseline_creat	aki
1	19008	FALSE	2.05	2019-07-08 14:37:33	NA	0
2	19008	FALSE	1.65	2019-07-09 08:37:33	NA	0
3	19008	FALSE	1.58	2019-07-29 08:37:33	NA	0
4	19008	FALSE	1.77	2019-08-10 08:37:33	NA	0
5	19008	FALSE	1.47	2019-09-25 02:37:33	NA	0
6	19008	FALSE	1.64	2019-11-25 14:37:33	NA	0

Actually, by default the toy dataset only has patient values $\pm$ 5 days from the admission date, and because the baseline creatinine value calculates using values from 365 to 7 days prior, you'll notice that the flagger reverts to the rolling window definition. This is important: in the absence of available baseline creatinine values, the flagger defaults to a rolling minimum comparison. Indeed, most of the checking for AKI occurs outside of period of hospitalization. Normally, of course, patients won't have times restricted to just $\pm$ 5 days, but this is a good opportunity to showcase one of the flagger features: the **eGFR-based imputation of baseline creatinine**.

The following equation is known as the CKD-EPI equation.

$$GFR = 141 \times min(S_{cr}/\kappa, 1)^{\alpha} \times max(S_{cr}/\kappa, 1)^{-1.209} \times 0.993^{Age} \times (1 + 0.018f) \times (1 + 0.159b) \quad (3.1)$$

where:

- GFR ($\frac{mL/min}{1.73m^2}$) is the glomerular filtration rate
- S_{cr} ($\frac{mg}{dL}$) is the serum creatinine
- κ (unitless) is 0.7 for females and 0.9 for males
- α (unitless) is -0.329 for females and -0.411 for males
- f is 1 if female, 0 if male
- b is 1 if black, 0 if another race

The idea is as follows: based on the above equation, we assume a GFR of 75 and then use the age, sex, and race to determine an estimate for the baseline creatinine. Theory aside, simply pass `eGFR_impute = True` into the flagger and this will add values where the patient was missing outpatient values 365 to 7 days prior to admission.

Python

Note: The toy dataset doesn't come with demographic information by default, but simply passing `include_demographic_info = True` adds in the age, race, and sex columns. We need to specify that sex is female & race is black in the flagger as well.

```
toy = generate_toy_data(num_patients=100, include_demographic_info = True)

toy.head()
```

	patient_id	age	female	black	inpatient	time	creatinine
0	12732	64.5	True	True	False	2020-02-23 23:42:42	1.45
1	12732	64.5	True	True	False	2020-02-24 05:42:42	1.59
2	12732	64.5	True	True	True	2020-02-28 05:42:42	1.46
3	12732	64.5	True	True	True	2020-03-01 05:42:42	1.51
4	12732	64.5	True	True	True	2020-03-01 23:42:42	1.52

```
flagger = AKIFlagger(HB_trumping = True, eGFR_impute = True, add_baseline_creat =␣
↪True,
                     sex = 'female', race = 'black')

out = flagger.returnAKIpatients(toy)

out.head()
```

patient_id	time	age	female	black	inpatient	creatinine	baseline_creat	aki
12732	2020-02-23 23:42:42	64.5	True	True	False	1.45	0.9300765849293293	0
12732	2020-02-24 05:42:42	64.5	True	True	False	1.59	0.9300765849293293	0
12732	2020-02-28 05:42:42	64.5	True	True	True	1.46	0.9300765849293293	1
12732	2020-03-01 05:42:42	64.5	True	True	True	1.51	0.9300765849293293	1
12732	2020-03-01 23:42:42	64.5	True	True	True	1.52	0.9300765849293293	1

R

There are actually two toy datasets that come with the packages: `toy` and `toy.demo`. `toy.demo` is the toy dataframe with columns for age, sex, and race. As such, all we have to do is run

```
out <- returnAKIpatients(toy.demo, HB_trumping = T, eGFR_impute = T)

head(out)
```

	patient_id	inpatient	creatinine	time	age	sex	race	baseline_creat	aki
1	19008	FALSE	1.94	2019-12-30 02:37:33	52.9	TRUE	FALSE	0.8806117024042	0
2	19008	TRUE	1.41	2020-01-02 02:37:33	52.9	TRUE	FALSE	0.8806117024042	1
3	19008	TRUE	1.2	2020-01-02 14:37:33	52.9	TRUE	FALSE	0.8806117024042	1
4	19008	TRUE	1.4	2020-01-03 02:37:33	52.9	TRUE	FALSE	0.8806117024042	1
5	19008	TRUE	1.49	2020-01-03 14:37:33	52.9	TRUE	FALSE	0.8806117024042	1
6	19008	TRUE	1.71	2020-01-03 20:37:33	52.9	TRUE	FALSE	0.8806117024042	1

That about does it for the basics! There are a slew of other features, some of which are listed in the *Additional Features* section. For a full listing of the features and appropriate use cases, see the *Documentation* at akiflagger.readthedocs.io.

CHAPTER 4

Additional Features and Common Use Cases

For most use cases, you will just need to specify *rolling-window* or *back-calculate* and the AKI-column will be returned. There are a slew of other features, some of which are listed below. For a full listing of the features and appropriate use cases, see the *Documentation* at akiflagger.readthedocs.io.

4.1 → Adding padding to the rolling window

It's often the case that you want to add some padding to the window to account for variations occurring on the floor (52 hour & 172 hour windows instead, for example). If the amount of padding you would like to add is the same for both the smaller and larger window, simply pass `padding='_hours'` filling the blank with the number of hours to add to the windows. If the pad times are different between windows, the parameters `pad1time` and `pad2time` allow you to add just this padding to the initial windows of 48 and 172 hours. In fact, if you wanted a window of 36 hours, you could even set *pad1time = '-12hours'*; this is one way in which you could modify the rolling window.

Python

```
# Example 0: Adding 4-hour padding to windows

flagger = AKIFlagger(padding = '4hours')

example0 = flagger.returnAKIpatients(toy)

example0[example0.aki > 0].head(3)
```

patient_id	time	inpatient	creatinine	aki
12732	2020-02-24 23:42:42	False	1.61	1
19845	2020-05-12 18:02:54	True	0.76	2
19845	2020-05-14 18:02:54	True	0.89	2

R

```
# Example 0: Adding 4-hour padding to windows

example0 <- returnAKIpatients(toy, padding = as.difftime(4, units = 'hours'))

head(example0[example0$aki > 0])
```

	patient_id	inpatient	creatinine	time	aki
1	19008	FALSE	2.06	2019-11-26 08:37:33	1
2	13264	FALSE	0.75	2019-08-11 17:16:57	3
3	13264	FALSE	0.5	2019-11-07 05:16:57	1
4	13264	FALSE	0.63	2019-11-08 23:16:57	2
5	18752	FALSE	1.18	2019-09-13 01:18:00	1
6	10537	FALSE	1.34	2019-11-08 07:55:12	1

4.2 → Working with different column names

Python

As an additional example, the patient identifier will often come in as *'PAT_MRN_ID'* or *'PAT_ENC_CSN_ID'* (or something of the sort) if it is coming from a typical clinical data warehouse/repository. Accordingly, these should be passed in as options to the flagger.

```
# Example 1: Working with different column names

dataframe = toy.rename(columns = {'patient_id': 'PAT_MRN_ID', 'creatinine':'CREATININE
↪', 'inpatient': 'INPATIENT', 'time': 'TIME'
                                  'age': 'AGE', 'female': 'SEX', 'black': 'RACE'})

flagger = AKIFlagger(patient_id = 'PAT_MRN_ID', inpatient = 'INPATIENT', time = 'TIME
↪', creatinine = 'CREATININE', age = 'AGE', sex = 'SEX', race = 'RACE')

example1 = flagger.returnAKIpatients(dataframe)

example1.head(3)
```

PAT_MRN_ID	TIME	AGE	SEX	RACE	INPATIENT	CREATININE	aki
12732	2020-02-22 11:42:42	64.5	True	True	False	1.62	0
12732	2020-02-23 11:42:42	64.5	True	True	False	1.52	0
12732	2020-02-24 23:42:42	64.5	True	True	False	1.63	0

R

Say we had a dataframe which looked like this:

	PAT_MRN_ID	OUTPATIENT	TIME	CREATININE
1	12732	TRUE	2019-11-16 05:42:42	1.05
2	12732	TRUE	2019-11-20 05:42:42	1.61
3	12732	TRUE	2020-01-15 05:42:42	1.42

In order to pass it to the flagger, we need to shape our data in a way that the flagger will understand. This means converting the outpatient columns to inpatient, and specifying the names of the columns as follows

```r
# Example 1: Working with different column names

library(dplyr) # rename function from dplyr library

dataframe$OUTPATIENT <- !dataframe$OUTPATIENT # turn the dataframe into inpatient␣
↪instead of outpatient by logically inverting it

dataframe <- dataframe %>% rename('patient_id' = 'PAT_MRN_ID', 'inpatient' =
↪'OUTPATIENT', 'time' = 'TIME', 'creatinine' = 'CREATININE')

head(returnAKIpatients(dataframe), n = 3L)
```

	patient_id	inpatient	creatinine	time	aki
1	19008	FALSE	2.05	2019-07-08 14:37:33	0
2	19008	FALSE	1.65	2019-07-09 08:37:33	0
3	19008	FALSE	1.58	2019-07-29 08:37:33	0

4.3 → Adding in rolling-window minimum creatinines

To add in the baseline creatinine, simply pass the flag `add_min_creat = True` to the flagger. This will add in two columns which contain the minimum values in the rolling window, which is an intermediate column generated to calculate AKI; the flag adds in the column which the current creatinine is checked against.

Python

```python
# Example 2: Adding in rolling-window minima

flagger = AKIFlagger(add_min_creat = True)

example2 = flagger.returnAKIpatients(toy)

example2.head(3)
```

patient_id	time	inpatient	creatinine	min_creat48	min_creat168	aki
12732	2020-02-22 17:42:42	False	1.05	1.05	1.05	0
12732	2020-02-26 05:42:42	False	1.26	1.26	1.05	0
12732	2020-02-29 05:42:42	True	1.06	1.06	1.05	0

R

```r
# Example 2: Adding in rolling-window minima

example2 <- returnAKIpatients(toy, add_min_creat = T)

head(example2)
```

	patient_id	inpatient	creatinine	time	min_creat48	min_creat7d	aki
1	19008	FALSE	2.05	2019-07-08 14:37:33	2.05	2.05	0
2	19008	FALSE	1.65	2019-07-09 08:37:33	1.65	1.65	0
3	19008	FALSE	1.58	2019-07-29 08:37:33	1.58	1.58	0
4	19008	FALSE	1.77	2019-08-10 08:37:33	1.77	1.77	0
5	19008	FALSE	1.47	2019-09-25 02:37:33	1.47	1.47	0
6	19008	FALSE	1.64	2019-11-25 14:37:33	1.64	1.64	0

4.4 → Adding in baseline creatinine

To add in the baseline creatinine, simply pass the flag `add_baseline_creat = True` to the flagger. Note that the baseline creatinine is not defined for outpatient measurements. Baseline creatinine can be thought of as the "resting" creatinine before coming into the hospital, so it doesn't make much sense to define the baseline creatinine outside of a hospital visit.

Python

```
# Example 3: Adding in baseline creatinine

toy = generate_toy_data(include_demographic_info = True)

flagger = AKIFlagger(HB_trumping = True, eGFR_impute = True, #Specifying both␣
↪calculation methods
                     add_baseline_creat = True, # Additional parameter to add in␣
↪baseline creatinine values
                     age = 'age', sex = 'female', race = 'black')

example3 = flagger.returnAKIpatients(toy)

example3[~example3.baseline_creat.isnull()].head(3)
```

patient_id	time	age	female	black	inpatient	creatinine	baseline_creat	aki
12732	2020-02-22 11:42:42	64.5	True	True	False	1.62	0.9300765849293293	0
12732	2020-02-25 11:42:42	64.5	True	True	False	1.63	0.9300765849293293	0
12732	2020-02-25 17:42:42	64.5	True	True	False	1.52	0.9300765849293293	0

R

```
# Example 3: Adding in baseline creatinine

example3 <- returnAKIpatients(toy, add_baseline_creat = T)

head(example3)
```

	patient_id	inpatient	creatinine	time	age	sex	race	baseline_creat	aki
1	19008	FALSE	1.94	2019-12-30 02:37:33	52.9	TRUE	FALSE	0.8806117024042	0
2	19008	TRUE	1.41	2020-01-02 02:37:33	52.9	TRUE	FALSE	0.8806117024042	1
3	19008	TRUE	1.2	2020-01-02 14:37:33	52.9	TRUE	FALSE	0.8806117024042	1
4	19008	TRUE	1.4	2020-01-03 02:37:33	52.9	TRUE	FALSE	0.8806117024042	1
5	19008	TRUE	1.49	2020-01-03 14:37:33	52.9	TRUE	FALSE	0.8806117024042	1
6	19008	TRUE	1.71	2020-01-03 20:37:33	52.9	TRUE	FALSE	0.8806117024042	1

CHAPTER 5

More information

For more information on the package, feel free to contact francis.p.wilson@yale.edu or is439@yale.edu.

Useful guides exist for more information about AKI, rolling windows, the back-calculation imputation method.

- AKI
- KDIGO guidelines
- KDIGO standard definitions
- Rolling window
- Back-calculation
- CKD-EPI equation

The source code for the package can be found on GitHub.

- genindex

CHAPTER 6

Introduction

Acute Kidney Injury (AKI) is a sudden onset of kidney failure and damage marked by an increase in the serum creatinine levels (amongst other biomarkers) of the patient. Kidney Disease Improving Global Outcomes (KDIGO) has a set of guidelines and standard definitions of AKI:

- *Stage 1:* 50% increase in creatinine in < 7 days or 0.3 increase in creatinine in < 48 hours
- *Stage 2:* 100% increase in (or doubling of) creatinine in < 7 days
- *Stage 3:* 200% increase in (or tripling of) creatinine in < 7 days

This package contains a flagger to determine if a patient has developed AKI based on the criterion above. More information about the specific data input format and examples can be found in the *Getting started* section.

CHAPTER 7

Installation

Python

You can install the flagger with `pip`. Simply type the following into command line and the package should install properly.

```
pip install akiFlagger
```

To ensure that it is working properly, you can open a Python session and test it with

```
import akiFlagger

akiFlagger.__version__

>> '1.0.0'
```

R

You can install the flagger through CRAN. Simply type the following into an RStudio terminal and the package should install properly.

```
install.packages('akiFlagger')
```

To ensure that it is working properly, you can open an RStudio session and test it with

```
library(akiFlagger)
```

CHAPTER 8

Getting started

This package is meant to handle patient data. Let's walk through an example of how to use this package with some toy data since real patient data is probably protected health information.

Once you've installed the package following the instructions in *Installation*, you're ready to get started. To begin with, we'll import the `akiFlagger` module.

Python

```
import akiFlagger

print(akiFlagger.__version__)

from akiFlagger import AKIFlagger, generate_toy_data

>> '1.0.0'
```

R

```
library(akiFlagger)

?returnAKIpatients

>  Rendering development documentation for 'returnAKIpatients'
```

8.1 Let's start off by creating some toy data.

Python

The flagger comes with a built-in generator of a toy dataset to demonstrate how it works. Simply call the *generate_toy_data()* function. By default, the toy dataset has 100 patients, but let's initialize ours with 1000 patients.

```
toy = generate_toy_data(num_patients=1000)

print('Toy dataset shape: {}'.format(toy.shape))

>> Successfully generated toy data!

   Toy dataset shape: (9094, 6)
```

The toy dataset comes with columns for the patient identifier, the encounter identifier, whether the measurement was inpatient or outpatient, the creatinine measurement and time at which the measurement was taken. `toy.head()` should yield something like this:

	patient_id	inpatient	time	creatinine
0	12732	False	2020-02-23 23:42:42	1.06
1	12732	False	2020-02-24 23:42:42	1.26
2	12732	False	2020-02-27 05:42:42	1.05
3	12732	True	2020-03-01 17:42:42	1.42
4	12732	True	2020-03-03 05:42:42	1.61

R

The R package comes with a built-in dataset, *toy*. The toy dataset comes with columns for the patient identifier, inpatient, the creatinine measurement and the time at which the measurement was taken. `head(toy)` should yield something like this:

	patient_id	inpatient	time	creatinine
1	12732	FALSE	2019-11-16 05:42:42	1.05
2	12732	FALSE	2019-11-20 05:42:42	1.61
3	12732	FALSE	2020-01-15 05:42:42	1.42
4	12732	FALSE	2020-02-27 11:42:42	1.26
5	12732	TRUE	2020-03-01 17:42:42	1.06
6	19845	FALSE	2019-11-20 18:02:54	0.89

Tip!

In order to calculate AKI, the flagger expects a dataset with certain columns in it. Depending on the type of computation you are interested in, your dataset will need to have different columns. Here's a brief rundown of the necessary columns.

- *Rolling-window*: **patient_id**, **inpatient**, **time**, and **creatinine**
- *Back-calculate*: **patient_id**, **inpatient**, **time**, and **creatinine**
- *eGFR-imputed baseline creatinine*: **age**, **sex** (defaults to female), and **race** (defaults to black).

By default, the naming system is as follows:

patient_id → **'patient_id'**

inpatient/outpatient → **'inpatient'**

creatinine →**'creatinine'**

time → **'time'**

If you have different names for your columns, you **must specify them.**

8.2 Example: Rolling-window

The next code block runs the flagger and returns those patients who satisfy the AKI conditions according to the KDIGO guidelines for change in creatinine values by the rolling-window definition, categorized as follows:

Stage 1: **(1)** 50% ↑ in creatinine in < 7 days OR **(2)** 0.3 mg/dL ↑ in creatinine in < 48 hours

Stage 2: 100% ↑ (or doubling of) in creatinine in < 7 days

Stage 3: 200% ↑ (or tripling of) in creatinine in < 7 days

Python

```
flagger = AKIFlagger()

out = flagger.returnAKIpatients(toy)

out = out.reset_index() # By default, the returned output has the patient_id and time␣
↪as hierarchical indices

out.head()
```

We can take a look at what our dataframe looks like. `out.head()` yields this:

patient_id	time	inpatient	creatinine	aki
12732	2020-02-23 17:42:42	False	1.42	0
12732	2020-02-28 17:42:42	True	1.26	0
12732	2020-02-29 23:42:42	True	1.05	0
12732	2020-03-02 17:42:42	True	1.61	1
19845	2020-05-08 00:02:54	False	0.76	0

Notice that the dataframe looks exactly the same as we inputted into the flagger save an extra column added, *aki*. This column has values of either 0, 1, 2, or 3, depending on which stage AKI the flagger found. The flagger runs on a row-wise basis, meaning that each row is checked for the increase in creatinine. Should, for example, a patient meet the criterion multiple times within a single encounter, the flagger will flag each measurement as a case of AKI.

Warning: The column names specified within the flagger should match the dataset exactly. The full list of acceptable names can be found in the *returnAKIpatients()* function in the genindex section. For certain cases, the flagger understands special names. For example, *sex = 'male'* will autoconvert the sex column from female to male. But you still need to have a column named *male* in your data frame, otherwise an error will occur.

We can take a look at what the flagger flagged as AKI. `out[out.aki > 0].head()` should give a list of some patients which were flagged. From that, we can subset the dataset on any given patient:

```
out[out.aki > 0].head() # this will give the rows which were marked as AKI by the␣
↪flagger
out[out.patient_id == 19845] # from that, we can find which patients were flagged␣
↪with AKI
```

	patient_id	time	inpatient	creatinine	aki
4	19845	2020-05-08 00:02:54	False	0.76	0
5	19845	2020-05-08 06:02:54	False	0.89	0
6	19845	2020-05-09 18:02:54	False	1.07	1
7	19845	2020-05-12 18:02:54	True	0.43	0
8	19845	2020-05-13 18:02:54	True	0.34	0
9	19845	2020-05-14 18:02:54	True	1.12	3

Notice how as we would expect, when the creatinine more than tripled from 0.34 to 1.12, the flagger correctly identified it as Stage 3 AKI.

You can even look at aggregate counts if you wanted as follows (but don't take the numbers too seriously, of course, because this is toy data):

```
aki_counts = out.aki.value_counts()
print('AKI counts')
print('----------')
print('No AKI: {}\nStage 1: {}\nStage 2: {}\nStage 3: {}'.format(aki_counts[0], aki_
↪counts[1], aki_counts[2], aki_counts[3]))

>>  AKI counts
    ----------
    No AKI: 571
    Stage 1: 211
    Stage 2: 99
    Stage 3: 70
```

You can play around with the output of the `returnAKIpatients()` function in-depth to get a better understanding of how the flagger is operating. There are even optional parameters such as `add_min_creat = True` within the flagger which includes some of the intermediate steps the flagger is generating along to calculate AKI. Next, we'll take a look at an example of the other AKI-calculation method, the back-calculation method.

R

```
library(akiFlagger)

out <- returnAKIpatients(toy)

head(out)
```

We can take a look at what the flagger returns. `head(out)` should return:

	patient_id	inpatient	creatinine	time	aki
1	19008	FALSE	2.05	2019-07-08 14:37:33	0
2	19008	FALSE	1.65	2019-07-09 08:37:33	0
3	19008	FALSE	1.58	2019-07-29 08:37:33	0
4	19008	FALSE	1.77	2019-08-10 08:37:33	0
5	19008	FALSE	1.47	2019-09-25 02:37:33	0
6	19008	FALSE	1.64	2019-11-25 14:37:33	0

Notice that the dataframe looks exactly the same as we inputted into the flagger save an extra column added, *aki*. This column has values of either 0, 1, 2, or 3, depending on which stage AKI the flagger found. The flagger runs on a row-wise basis, meaning that each row is checked for the increase in creatinine. Should, for example, a patient meet the criterion multiple times within a single encounter, the flagger will flag each measurement as a case of AKI.

Warning: The patient dataset you input should have minimally these columns: `patient_id`, `inpatient`, `time`, and `creatinine`. If you are interested in demographic-based imputation, you'll also want to include the columns `age`, `sex`, and `race`.

We can take a look at what the flagger flagged as AKI. `head(out[out$aki > 0])` should give a list of some patients which were flagged. From that, we can subset the dataset on any given patient:

```
head(out[out$aki > 0])

out[out$patient_id == 13264]
```

	patient_id	inpatient	creatinine	time	aki
1	13264	FALSE	0.47	2019-07-22 23:16:57	0
2	13264	FALSE	0.1	2019-08-06 23:16:57	0
3	13264	FALSE	0.75	2019-08-11 17:16:57	3
4	13264	FALSE	0.79	2019-08-23 11:16:57	0
5	13264	FALSE	0.61	2019-09-02 17:16:57	0
6	13264	FALSE	0.59	2019-09-03 05:16:57	0
7	13264	FALSE	0.55	2019-09-19 05:16:57	0
8	13264	FALSE	0.49	2019-10-04 17:16:57	0
9	13264	FALSE	0.18	2019-10-09 23:16:57	0
10	13264	FALSE	0.27	2019-11-02 17:16:57	0
11	13264	FALSE	0.5	2019-11-07 05:16:57	1
12	13264	FALSE	0.63	2019-11-08 23:16:57	2
13	13264	FALSE	0.29	2019-11-12 05:16:57	0
14	13264	FALSE	0.22	2019-12-15 11:16:57	0
15	13264	TRUE	0.28	2020-01-12 05:16:57	0

Notice how as we would expect, when the creatinine more than tripled from 0.1 to 0.72, the flagger correctly identified it as Stage 3 AKI. Additionally, row 11 was flagged as stage 1 because that was a greater than 50% increase from 0.27 and row 12 was flagged because it was a greater than 100% increase from 0.27. Even though the flagger is performing a row-wise computation, it is comparing the current creatinine value with the minimum in the past `window1` hours (defaults to 48 hours).

You can look at aggregate counts if you wanted as follows (but don't take the numbers too seriously, of course, because this is toy data):

```
table(out$aki)

>>    0    1    2    3
    1001   44   19   14
```

8.3 Example: Back-calculation

Next, we'll run the flagger to "back-calculate" AKI; that is, using the **median outpatient creatinine values from 365 to 7 days prior to admission** to impute a baseline creatinine value. Then, we'll run the same KDIGO criterion (except for the 0.3 increase) comparing the creatinine value to baseline creatinine.

Python

```
flagger = AKIFlagger(HB_trumping = True, add_baseline_creat = True)

out = flagger.returnAKIpatients(toy)

out.head()
```

patient_id	time	inpatient	creatinine	baseline_creat	aki
12732	2020-02-22 23:42:42	False	1.26		0
12732	2020-02-24 05:42:42	False	1.61		1
12732	2020-02-24 23:42:42	False	1.05		0
12732	2020-02-26 23:42:42	False	1.42		1
12732	2020-03-03 11:42:42	True	1.06		0

R

```
out <- returnAKIpatients(toy, HB_trumping = T, add_baseline_creat = T)

head(out)
```

	patient_id	inpatient	creatinine	time	baseline_creat	aki
1	19008	FALSE	2.05	2019-07-08 14:37:33	NA	0
2	19008	FALSE	1.65	2019-07-09 08:37:33	NA	0
3	19008	FALSE	1.58	2019-07-29 08:37:33	NA	0
4	19008	FALSE	1.77	2019-08-10 08:37:33	NA	0
5	19008	FALSE	1.47	2019-09-25 02:37:33	NA	0
6	19008	FALSE	1.64	2019-11-25 14:37:33	NA	0

Actually, by default the toy dataset only has patient values $\pm$ 5 days from the admission date, and because the baseline creatinine value calculates using values from 365 to 7 days prior, you'll notice that the flagger reverts to the rolling window definition. This is important: in the absence of available baseline creatinine values, the flagger defaults to a rolling minimum comparison. Indeed, most of the checking for AKI occurs outside of period of hospitalization. Normally, of course, patients won't have times restricted to just $\pm$ 5 days, but this is a good opportunity to showcase one of the flagger features: the **eGFR-based imputation of baseline creatinine**.

The following equation is known as the CKD-EPI equation.

$$GFR = 141 \times min(S_{cr}/\kappa, 1)^{\alpha} \times max(S_{cr}/\kappa, 1)^{-1.209} \times 0.993^{Age} \times (1 + 0.018f) \times (1 + 0.159b) \quad (8.1)$$

where:

- GFR ($\frac{mL/min}{1.73m^2}$) is the glomerular filtration rate
- S_{cr} ($\frac{mg}{dL}$) is the serum creatinine
- κ (unitless) is 0.7 for females and 0.9 for males
- α (unitless) is -0.329 for females and -0.411 for males
- f is 1 if female, 0 if male
- b is 1 if black, 0 if another race

The idea is as follows: based on the above equation, we assume a GFR of 75 and then use the age, sex, and race to determine an estimate for the baseline creatinine. Theory aside, simply pass `eGFR_impute = True` into the flagger and this will add values where the patient was missing outpatient values 365 to 7 days prior to admission.

Python

Note: The toy dataset doesn't come with demographic information by default, but simply passing `include_demographic_info = True` adds in the age, race, and sex columns. We need to specify that sex is female & race is black in the flagger as well.

```
toy = generate_toy_data(num_patients=100, include_demographic_info = True)

toy.head()
```

	patient_id	age	female	black	inpatient	time	creatinine
0	12732	64.5	True	True	False	2020-02-23 23:42:42	1.45
1	12732	64.5	True	True	False	2020-02-24 05:42:42	1.59
2	12732	64.5	True	True	True	2020-02-28 05:42:42	1.46
3	12732	64.5	True	True	True	2020-03-01 05:42:42	1.51
4	12732	64.5	True	True	True	2020-03-01 23:42:42	1.52

```
flagger = AKIFlagger(HB_trumping = True, eGFR_impute = True, add_baseline_creat =␣
↪True,
                     sex = 'female', race = 'black')

out = flagger.returnAKIpatients(toy)

out.head()
```

patient_id	time	age	female	black	inpatient	creatinine	baseline_creat	aki
12732	2020-02-23 23:42:42	64.5	True	True	False	1.45	0.9300765849293293	0
12732	2020-02-24 05:42:42	64.5	True	True	False	1.59	0.9300765849293293	0
12732	2020-02-28 05:42:42	64.5	True	True	True	1.46	0.9300765849293293	1
12732	2020-03-01 05:42:42	64.5	True	True	True	1.51	0.9300765849293293	1
12732	2020-03-01 23:42:42	64.5	True	True	True	1.52	0.9300765849293293	1

R

There are actually two toy datasets that come with the packages: `toy` and `toy.demo`. `toy.demo` is the toy dataframe with columns for age, sex, and race. As such, all we have to do is run

```
out <- returnAKIpatients(toy.demo, HB_trumping = T, eGFR_impute = T)

head(out)
```

	patient_id	inpatient	creatinine	time	age	sex	race	baseline_creat	aki
1	19008	FALSE	1.94	2019-12-30 02:37:33	52.9	TRUE	FALSE	0.8806117024042	0
2	19008	TRUE	1.41	2020-01-02 02:37:33	52.9	TRUE	FALSE	0.8806117024042	1
3	19008	TRUE	1.2	2020-01-02 14:37:33	52.9	TRUE	FALSE	0.8806117024042	1
4	19008	TRUE	1.4	2020-01-03 02:37:33	52.9	TRUE	FALSE	0.8806117024042	1
5	19008	TRUE	1.49	2020-01-03 14:37:33	52.9	TRUE	FALSE	0.8806117024042	1
6	19008	TRUE	1.71	2020-01-03 20:37:33	52.9	TRUE	FALSE	0.8806117024042	1

That about does it for the basics! There are a slew of other features, some of which are listed in the *Additional Features* section. For a full listing of the features and appropriate use cases, see the *Documentation* at akiflagger.readthedocs.io.

CHAPTER 9

Using the GUI

CHAPTER 10

Additional Features and Common Use Cases

For most use cases, you will just need to specify *rolling-window* or *back-calculate* and the AKI-column will be returned. There are a slew of other features, some of which are listed below. For a full listing of the features and appropriate use cases, see the *Documentation* at akiflagger.readthedocs.io.

→ **Adding padding to the rolling window**

It's often the case that you want to add some padding to the window to account for variations occurring on the floor (52 hour & 172 hour windows instead, for example). If the amount of padding you would like to add is the same for both the smaller and larger window, simply pass `padding='_hours'` filling the blank with the number of hours to add to the windows. If the pad times are different between windows, the parameters `pad1time` and `pad2time` allow you to add just this padding to the initial windows of 48 and 172 hours. In fact, if you wanted a window of 36 hours, you could even set *pad1time = '-12hours'*; this is one way in which you could modify the rolling window.

Python

```
# Example 0: Adding 4-hour padding to windows

flagger = AKIFlagger(padding = '4hours')

example0 = flagger.returnAKIpatients(toy)

example0[example0.aki > 0].head(3)
```

patient_id	time	inpatient	creatinine	aki
12732	2020-02-24 23:42:42	False	1.61	1
19845	2020-05-12 18:02:54	True	0.76	2
19845	2020-05-14 18:02:54	True	0.89	2

R

```
# Example 0: Adding 4-hour padding to windows

example0 <- returnAKIpatients(toy, padding = as.difftime(4, units = 'hours'))
```

(continues on next page)

(continued from previous page)

```
head(example0[example0$aki > 0])
```

	patient_id	inpatient	creatinine	time	aki
1	19008	FALSE	2.06	2019-11-26 08:37:33	1
2	13264	FALSE	0.75	2019-08-11 17:16:57	3
3	13264	FALSE	0.5	2019-11-07 05:16:57	1
4	13264	FALSE	0.63	2019-11-08 23:16:57	2
5	18752	FALSE	1.18	2019-09-13 01:18:00	1
6	10537	FALSE	1.34	2019-11-08 07:55:12	1

→ Working with different column names

Python

As an additional example, the patient identifier will often come in as *'PAT_MRN_ID'* or *'PAT_ENC_CSN_ID'* (or something of the sort) if it is coming from a typical clinical data warehouse/repository. Accordingly, these should be passed in as options to the flagger.

```
# Example 1: Working with different column names

dataframe = toy.rename(columns = {'patient_id': 'PAT_MRN_ID', 'creatinine':'CREATININE
↪', 'inpatient': 'INPATIENT', 'time': 'TIME'
                                  'age': 'AGE', 'female': 'SEX', 'black': 'RACE'})

flagger = AKIFlagger(patient_id = 'PAT_MRN_ID', inpatient = 'INPATIENT', time = 'TIME
↪', creatinine = 'CREATININE', age = 'AGE', sex = 'SEX', race = 'RACE')

example1 = flagger.returnAKIpatients(dataframe)

example1.head(3)
```

PAT_MRN_ID	TIME	AGE	SEX	RACE	INPATIENT	CREATININE	aki
12732	2020-02-22 11:42:42	64.5	True	True	False	1.62	0
12732	2020-02-23 11:42:42	64.5	True	True	False	1.52	0
12732	2020-02-24 23:42:42	64.5	True	True	False	1.63	0

R

Say we had a dataframe which looked like this:

	PAT_MRN_ID	OUTPATIENT	TIME	CREATININE
1	12732	TRUE	2019-11-16 05:42:42	1.05
2	12732	TRUE	2019-11-20 05:42:42	1.61
3	12732	TRUE	2020-01-15 05:42:42	1.42

In order to pass it to the flagger, we need to shape our data in a way that the flagger will understand. This means converting the outpatient columns to inpatient, and specifying the names of the columns as follows

```
# Example 1: Working with different column names

library(dplyr) # rename function from dplyr library
```

(continues on next page)

(continued from previous page)

```
dataframe$OUTPATIENT <- !dataframe$OUTPATIENT # turn the dataframe into inpatient␣
↪instead of outpatient by logically inverting it

dataframe <- dataframe %>% rename('patient_id' = 'PAT_MRN_ID', 'inpatient' =
↪'OUTPATIENT', 'time' = 'TIME', 'creatinine' = 'CREATININE')

head(returnAKIpatients(dataframe), n = 3L)
```

	patient_id	inpatient	creatinine	time	aki
1	19008	FALSE	2.05	2019-07-08 14:37:33	0
2	19008	FALSE	1.65	2019-07-09 08:37:33	0
3	19008	FALSE	1.58	2019-07-29 08:37:33	0

→ Adding in rolling-window minimum creatinines

To add in the baseline creatinine, simply pass the flag `add_min_creat = True` to the flagger. This will add in two columns which contain the minimum values in the rolling window, which is an intermediate column generated to calculate AKI; the flag adds in the column which the current creatinine is checked against.

Python

```
# Example 2: Adding in rolling-window minima

flagger = AKIFlagger(add_min_creat = True)

example2 = flagger.returnAKIpatients(toy)

example2.head(3)
```

patient_id	time	inpatient	creatinine	min_creat48	min_creat168	aki
12732	2020-02-22 17:42:42	False	1.05	1.05	1.05	0
12732	2020-02-26 05:42:42	False	1.26	1.26	1.05	0
12732	2020-02-29 05:42:42	True	1.06	1.06	1.05	0

R

```
# Example 2: Adding in rolling-window minima

example2 <- returnAKIpatients(toy, add_min_creat = T)

head(example2)
```

	patient_id	inpatient	creatinine	time	min_creat48	min_creat7d	aki
1	19008	FALSE	2.05	2019-07-08 14:37:33	2.05	2.05	0
2	19008	FALSE	1.65	2019-07-09 08:37:33	1.65	1.65	0
3	19008	FALSE	1.58	2019-07-29 08:37:33	1.58	1.58	0
4	19008	FALSE	1.77	2019-08-10 08:37:33	1.77	1.77	0
5	19008	FALSE	1.47	2019-09-25 02:37:33	1.47	1.47	0
6	19008	FALSE	1.64	2019-11-25 14:37:33	1.64	1.64	0

→ Adding in baseline creatinine

To add in the baseline creatinine, simply pass the flag `add_baseline_creat = True` to the flagger. Note that the baseline creatinine is not defined for outpatient measurements. Baseline creatinine can be thought of as the "resting" creatinine before coming into the hospital, so it doesn't make much sense to define the baseline creatinine outside of a hospital visit.

Python

```
# Example 3: Adding in baseline creatinine

toy = generate_toy_data(include_demographic_info = True)

flagger = AKIFlagger(HB_trumping = True, eGFR_impute = True, #Specifying both␣
↪calculation methods
                     add_baseline_creat = True, # Additional parameter to add in␣
↪baseline creatinine values
                     age = 'age', sex = 'female', race = 'black')

example3 = flagger.returnAKIpatients(toy)

example3[~example3.baseline_creat.isnull()].head(3)
```

patient_id	time	age	female	black	inpatient	creatinine	baseline_creat	aki
12732	2020-02-22 11:42:42	64.5	True	True	False	1.62	0.9300765849293293	0
12732	2020-02-25 11:42:42	64.5	True	True	False	1.63	0.9300765849293293	0
12732	2020-02-25 17:42:42	64.5	True	True	False	1.52	0.9300765849293293	0

R

```
# Example 3: Adding in baseline creatinine

example3 <- returnAKIpatients(toy, add_baseline_creat = T)

head(example3)
```

	patient_id	inpatient	creatinine	time	age	sex	race	baseline_creat	aki
1	19008	FALSE	1.94	2019-12-30 02:37:33	52.9	TRUE	FALSE	0.8806117024042	0
2	19008	TRUE	1.41	2020-01-02 02:37:33	52.9	TRUE	FALSE	0.8806117024042	1
3	19008	TRUE	1.2	2020-01-02 14:37:33	52.9	TRUE	FALSE	0.8806117024042	1
4	19008	TRUE	1.4	2020-01-03 02:37:33	52.9	TRUE	FALSE	0.8806117024042	1
5	19008	TRUE	1.49	2020-01-03 14:37:33	52.9	TRUE	FALSE	0.8806117024042	1
6	19008	TRUE	1.71	2020-01-03 20:37:33	52.9	TRUE	FALSE	0.8806117024042	1

CHAPTER 11

More information

For more information on the package, feel free to contact rishabh.p.saran@vanderbilt.edu or is439@yale.edu.

Useful guides exist for more information about AKI, rolling windows, the back-calculation imputation method.

- AKI
- KDIGO guidelines
- KDIGO standard definitions
- Rolling-window method
- Back-calculation method
- CKD-EPI equation

The source code for the package can be found on GitHub.

- genindex

Index