# WikiTopK

Calculate Top 25 pages for each domain by hour.

## Extra Features!

- Get *aggregated* Top 25 pages for a time interval
- Caches computed results

## How is caching handled?

Using luigi tasks parameterized by *start-time*, *end-time* and *aggregate* variables.
This ensures tasks that compute top 25 tasks are not duplicated for the same input.

## Installation

WikiTopK requires Pybuilder to install dependencies.

```
$ pip install pybuilder
```

### Install the dependencies.

```
$ pyb install_dependencies
```

## Command Line Interface

```
$ runapp.py -h
usage: runapp.py [-h] [-s [DATETIME]] [-e [ENDDATETIME]] [-a]

Compute Top 25 Wikipedia Pages for each sub domain

optional arguments:
  -h, --help            show this help message and exit
  -s [DATETIME], --datetime [DATETIME]
                        The Date and Time (Start) - format MM
-DD-YYYY-HH
  -e [ENDDATETIME], --enddatetime [ENDDATETIME]
                        The End Date and Time - format MM-DD-
YYYY-HH
  -a, --agg             Calculate Aggregated Top 25 between t
he interval

```

## Compute Top 25 for current hour

```
$ runapp.py
```

## Compute Top 25 for a datetime

```
$ runapp.py -s 06-11-2016-11
```

### Compute Top 25 for every hour between datetimes

```
$ runapp.py -s 06-11-2016-11 -e 06-13-2016-22
```

### Compute Aggregated Top 25 between datetimes

```
$ runapp.py -s 06-11-2016-11 -e 06-13-2016-22 -a
```

## Sample Runs

## - Compute Top 25 for a datetime

```
$ runapp.py -s 06-11-2016-11
Starting calculation with starttime: 11 Jun 2016 11 and endti
me: 11 Jun 2016 11 in aggregate mode False
[WikiTopK] - Fetching Hour Data for 2016-06-11 11:00:00
[WikiTopK] - Fetch Complete
[WikiTopK] - Computing Top 25 for 2016-06-11 11:00:00
[WikiTopK] - 5826481 entries read
[WikiTopK] - 8471 entries blacklisted
[WikiTopK] - 0 errorneous entries found
[WikiTopK] - Computing complete
[WikiTopK] - Top 25 computation complete
[WikiTopK] - Computed in 82.291s
```

## - Compute Top 25 for every hour between datetimes

```
$ runapp.py -s 06-11-2016-13 -e 06-11-2016-14
Starting calculation with starttime: 11 Jun 2016 13 and endti
me: 11 Jun 2016 14 in aggregate mode False
[WikiTopK] - Fetching Hour Data for 2016-06-11 14:00:00
[WikiTopK] - Fetch Complete
[WikiTopK] - Computing Top 25 for 2016-06-11 14:00:00
[WikiTopK] - 6484317 entries read
[WikiTopK] - 9318 entries blacklisted
[WikiTopK] - 0 errorneous entries found
[WikiTopK] - Computing complete
[WikiTopK] - Fetching Hour Data for 2016-06-11 13:00:00
[WikiTopK] - Fetch Complete
[WikiTopK] - Computing Top 25 for 2016-06-11 13:00:00
[WikiTopK] - 6266330 entries read
[WikiTopK] - 9051 entries blacklisted
[WikiTopK] - 0 errorneous entries found
[WikiTopK] - Computing complete
[WikiTopK] - Top 25 computation complete
[WikiTopK] - Computed in 185.81s
```

## - Compute Aggregated Top 25 between datetimes

```
$ runapp.py -s 06-11-2016-13 -e 06-11-2016-14 -a
Starting calculation with starttime: 11 Jun 2016 13 and endti
me: 11 Jun 2016 14 in aggregate mode True
[WikiTopK] - Fetching Hour Data for 2016-06-11 14:00:00
[WikiTopK] - Fetch Complete
[WikiTopK] - Fetching Hour Data for 2016-06-11 13:00:00
[WikiTopK] - Fetch Complete
[WikiTopK] - Aggregating and Computing Top 25 for interval 20
16-06-11 13:00:00 to 2016-06-11 14:00:00
[WikiTopK] - 6266330 entries read
[WikiTopK] - 9051 entries blacklisted
[WikiTopK] - 0 errorneous entries found
[WikiTopK] - 12750647 entries read
[WikiTopK] - 18369 entries blacklisted
[WikiTopK] - 0 errorneous entries found
[WikiTopK] - Computing complete
[WikiTopK] - Top 25 computation complete
[WikiTopK] - Computed in 162.175s
```

## What additional things would you want to operate this application in a production setting?

A cloud based storage location

A Monitoring system for application performance to visualize bottlenecks

Log Mangement and Failure Alert

Central Scheduler

## What might change about your solution if this

## application needed to run automatically for each hour of the day?

A Cron Job to automatically trigger the tasks

An Error handler since the tasks are under no supervision, the Error handler could either handle an error by adding it back in the queue or notifying an admin

## How would you test this application?

Functional Testing: Validation data can be obtained from ‘Pageview stats’ - a tool to analyze wikipedia page views. Though this tool only narrows down till a day, it provides filters like domain and blacklisted pages. Also to test the no duplication of task, you could run the task with overlapping intervals to verify if completed task is running again.

Unit Testing: Packages like mock(mockito) can be used for unit testing the application.

Performance Testing: The application is fairly slow. I profiled it to find the bottlenecks, which was largely due to downloading the data part and the creating the dictionary part. There could be enhancement in speed writing underlying dictionary creation or finding nlargest in Cython, but it is a trade-off with the convienience of using the pandas DataFrame.

## How you’d improve on this application design?

Assuming infinite resources. The main improvement would be leveraging distributed power. This application can be heavily distributed. Fetching data is an independent task that can be completely parallelized using any ETL framework. Similarly the function to find nlargest can also be parallelized per domain.

Apart from that, locally, it would speed up the application to transfer data between tasks without writing/reading from disk. Once the data was in memory, it would be faster to do computation completely before writing back into disk

## You can write your own workflow or use existing libraries (Luigi, Drake, etc).

I decided to use Luigi for workflow as it was very easy to ensure reruns of the same day or hour does not happen by parameterizing the tasks with the start and end time. Thus creating unique task ids for each input.