Text Visualization DRG Documentation

Release 0.1.0

Text Visualization DRG

October 30, 2016

Contents

This project is a prototype exploratory visual data analysis tool designed for social scientists working with large social message data sets.

Contents:

CHAPTER 1

Apps

1.1 Base

Base is the core app that serves the main pages of the visualization application. It also gathers together some miscellaneous utilities and shared classes that are used by other apps.

1.1.1 Views

class msgvis.apps.base.views.**LoginRequiredMixin**
A mixin that forces a login to view the CBTemplate.

class msgvis.apps.base.views.**HomeView**(***kwargs*)
The homepage view for the website.

class msgvis.apps.base.views.**ExplorerView**(***kwargs*)
The view for the visualization tool.

class msgvis.apps.base.views.**GrouperView**(***kwargs*)
The view for the visualization tool.

1.1.2 Context Processors

msgvis.apps.base.context_processors.**google_analytics**(*request*)
Adds a GOOGLE_ANALYTICS_ID variable to the template context.

Add this to your Django settings:

```
GOOGLE_ANALYTICS_ID = 'UA-XXXXXX-X'

TEMPLATE_CONTEXT_PROCESSORS += (
    'msgvis.apps.base.context_processors.google_analytics',
)
```

1.1.3 Template Tags

msgvis.apps.base.templatetags.active.**active**(*request*, *pattern*)
Checks the current request to see if it matches a pattern. If so, it returns 'active'.

To use, add this to your Django template:

```
{% load tags %}
<li class="{% active request home %}"><a href="/">Home</a></li>
```

1.1.4 Models

class msgvis.apps.base.models.**MappedValuesQuerySet**(**args*, ***kwargs*)
A special ValuesQuerySet that can re-map the dictionary keys while they are bing iterated over.

```
valuesQuerySet = queryset.values('some__ugly__field__expression')
mapped = MappedQuerySet.create_from(valuesQuerySet, {
    'some__ugly__field__expression': 'nice_expression'
})
mapped[0]
# { 'nice_expression': 5 }
```

classmethod **create_from**(*values_query_set*, *field_map*)
Create a MappedValueQuerySet with a field name mapping dictionary.

1.2 Importer

The Importer app is concerned with getting corpus data into the database. It defines a number of Django management commands for making this easier.

1.2.1 Commands

class msgvis.apps.importer.management.commands.import_corpus.**Command**
Import a corpus of message data into the database.

```
$ python manage.py import_corpus <file_path>
```

class msgvis.apps.importer.management.commands.import_twitter_languages.**Command**
Import supported languages from the Twitter API into the database. If the languages already exist in the database, they will not be duplicated.

Note: Requires the tweepy Twitter API library: `pip install tweepy`

Example:

```
$ python manage.py import_twitter_languages
```

class msgvis.apps.importer.management.commands.import_twitter_timezones.**Command**
Obtains a mapping of the Twitter-supported timezones from the Ruby on Rails TimeZone class.

Get the mapping dictionary from https://github.com/rails/rails/blob/master/activesupport/lib/active_support/values/time_zone.rb

Note: Requires Ruby on Rails to be installed: `gem install rails`.

Example:

```
$ python manage.py import_twitter_timezones setup/time_zone_mapping.rb
```

1.2.2 Twitter Integration

Utilities for working with Twitter.

msgvis.apps.importer.twitter.**tweepy_installed**()
Return True if tweepy is installed

msgvis.apps.importer.twitter.**get_tweepy_auth**()
Interactive commands for getting Twitter API authorization. Returns a tweepy.OAuthHandler.

msgvis.apps.importer.twitter.**get_languages**()
Get a list of languages supported by Twitter.

msgvis.apps.importer.twitter.**get_timezones**(*time_zones_mapping_file*)
Get a list of twitter-supported timezones as name/Olson code pairs.

1.2.3 Models

msgvis.apps.importer.models.**create_an_instance_from_json**(*json_str*, *dataset_obj*)
Given a dataset object, imports a tweet from json string into the dataset.

msgvis.apps.importer.models.**get_or_create_a_tweet_from_json_obj**(*tweet_data*, *dataset_obj*)
Given a dataset object, imports a tweet from json object into the dataset.

msgvis.apps.importer.models.**load_research_questions_from_json**(*json_str*)
Load research questions from json string

1.3 Enhance

The Enhance app is supposed to integrate with external tools to add metadata to the raw corpus data.

For example, it might use an external sentiment analysis tool to label messages for sentiment.

1.4 Dimensions

The Dimensions app provides functionality for asking about dimension metadata, including distributions within a dimension over a dataset.

1.4.1 Registry

Import this module to get access to dimension instances.

```
from msgvis.apps.dimensions import registry
time = registry.get_dimension('time') # returns a TimeDimension
time.get_distribution(a_dataset)
```

msgvis.apps.dimensions.registry.**register**(*dimensionClass*, *kwargs*)
Register a dimension

msgvis.apps.dimensions.registry.**get_dimension**(*dimension_key*)
Get a specific dimension by key

msgvis.apps.dimensions.registry.**get_dimensions**()
Get a list of all the registered dimensions.

msgvis.apps.dimensions.registry.**get_dimension_ids**()
Get a list of all the dimension keys.

1.4.2 Models

msgvis.apps.dimensions.models.**find_messages**(*queryset*)
If the given queryset is actually a *Dataset* model, get its messages queryset.

class msgvis.apps.dimensions.models.**CategoricalDimension**(*key*, *name=None*, *description=None*, *field_name=None*, *domain=None*)
A basic categorical dimension class.

Attributes:

- key (str): A string id for the dimension (e.g. 'time')
- name (str): A nicely-formatted name for the dimension (e.g. 'Number of Tweets')
- description (str): A longer explanation for the dimension (e.g. "The total number of tweets produced by this author.")
- **field_name (str): The name of the field in the database for this dimension (defaults to the key)**
  Related to the Message model: if you want sender name, use sender__name.

is_categorical()
Return True for real categorical dimensions

is_related_categorical()
Return True for real categorical dimensions

filter(*queryset*, ***kwargs*)
Apply a filter to a queryset and return the new queryset.

exclude(*queryset*, ***kwargs*)
Exclude some points from a queryset and return the new queryset.

group_by(*queryset*, *grouping_key=None*, *values_list=False*, *values_list_flat=False*, ***kwargs*)
Return a ValuesQuerySet that has been grouped by this dimension. The group value will be available as grouping_key in the dictionaries.

The grouping key defaults to the dimension key.

```
messages = dim.group_by(messages, 'value')
distribution = messages.annotate(count=Count('id'))
print distribution[0]
# { 'value': 'hello', 'count': 5 }
```

select_grouping_expression(*queryset*, *expression*)
Add an expression for grouping to the queryset's SELECT. Returns the queryset plus the alias for the expression.

For categorical dimensions this is a no-op. Beware if your expression refers to a related table!

get_domain(*queryset*, ***kwargs*)
Get the list of values of the dimension, either in natural order or sorted by frequency. The values will be drawn from the queryset.

get_domain_labels(*domain*)
Return a list of labels corresponding to the domain values

get_grouping_expression(*queryset*, ***kwargs*)
Given a set of messages (possibly filtered), returns a string that could be used with QuerySet.values() to group the messages by this dimension.

class msgvis.apps.dimensions.models.**ChoicesCategoricalDimension**(*key*, *name=None*, *description=None*, *field_name=None*, *domain=None*)
A categorical dimension where the values come from a choices set.

Don't use for related fields.

class msgvis.apps.dimensions.models.**RelatedCategoricalDimension**(*key*, *name=None*, *description=None*, *field_name=None*, *domain=None*)
A categorical dimension where the values are in a related table, e.g. sender name.

Currently doesn't really do much beyond CategoricalDimension.

is_related_categorical()
Return True for related categorical dimensions

class msgvis.apps.dimensions.models.**QuantitativeDimension**(*key*, *name=None*, *description=None*, *field_name=None*, *default_bins=50*, *min_bin_size=1*)
A generic quantitative dimension. This works for fields on Message or on related fields, e.g. field_name=sender__message_count

get_range(*queryset*)
Find a min and max for this dimension, as a tuple. If there isn't one, (None, None) is returned.

get_grouping_expression(*queryset*, *bins=None*, *bin_size=None*, ***kwargs*)
Generate a SQL expression for grouping this dimension. If you already know the bin size you want, you may provide it. Or the number of bins.

select_grouping_expression(*queryset*, *expression*)
Add an expression for grouping to the queryset's SELECT.

Returns a queryset, grouping_key tuple. The grouping_key could be used in values to identify the grouping expression.

group_by(*queryset*, *grouping_key=None*, *bins=None*, *bin_size=None*, ***kwargs*)
Return a ValuesQuerySet that has been grouped by this dimension. The group value will be available as grouping_key in the dictionaries.

The grouping key defaults to the dimension key.

If num_bins or bin_size is not provided, an estimate will be used.

```
messages = dim.group_by(messages, 'value', 100)
distribution = messages.annotate(count=Count('id'))
print distribution[0]
# { 'value': 'hello', 'count': 5 }
```

class msgvis.apps.dimensions.models.**RelatedQuantitativeDimension**(*key*, *name=None*, *description=None*, *field_name=None*, *default_bins=50*, *min_bin_size=1*)
A quantitative dimension on a related model, e.g. sender message count.

class msgvis.apps.dimensions.models.**TimeDimension**(*key*, *name=None*, *description=None*, *field_name=None*, *default_bins=50*, *min_bin_size=1*)
A dimension for time fields on Message

class msgvis.apps.dimensions.models.**TextDimension**(*key*, *name=None*, *description=None*, *field_name=None*, *domain=None*)
A dimension based on the words in a text field.

is_related_categorical()
Return True for related categorical dimensions

class msgvis.apps.dimensions.models.**DimensionKey**(**args*, ***kwargs*)
Dimension names for research questions.

key = None
The id of the dimension

1.5 Corpus

The Corpus app is concerned with the representation of raw message data and its associated metadata.

1.5.1 Models

class msgvis.apps.corpus.models.**Dataset**(**args*, ***kwargs*)
A top-level dataset object containing messages.

name = None
The name of the dataset

description = None
A description of the dataset.

created_at = None
The datetime.datetime when the dataset was created.

start_time = None
The time of the first real message in the dataset

end_time = None
The time of the last real message in the dataset

get_example_messages(*filters=[]*, *excludes=[]*)
Get example messages given some filters (dictionaries containing dimensions and filter params)

class msgvis.apps.corpus.models.**MessageType**(**args*, ***kwargs*)
The type of a message, e.g. retweet, reply, original, system...

name = None
The name of the message type

class msgvis.apps.corpus.models.**Language**(**args*, ***kwargs*)
Represents the language of a message or a user

code = None
A short language code like 'en'

name = None
The full name of the language

class msgvis.apps.corpus.models.**Url**(**args*, ***kwargs*)
A url from a message

domain = None
The root domain of the url

short_url = None
A shortened url

full_url = None
The full url

class msgvis.apps.corpus.models.**Hashtag**(**args*, ***kwargs*)
A hashtag in a message

text = None
The text of the hashtag, without the hash

class msgvis.apps.corpus.models.**Media**(**args*, ***kwargs*)
Linked media, e.g. photos or videos.

type = None
The kind of media this is.

media_url = None
A url where the media may be accessed

class msgvis.apps.corpus.models.**Timezone**(**args*, ***kwargs*)
The timezone of a message or user

olson_code = None
The timezone code from pytz.

name = None
Another name for the timezone, perhaps the country where it is located?

class msgvis.apps.corpus.models.**Person**(**args*, ***kwargs*)
A person who sends messages in a dataset.

dataset
Which *Dataset* this person belongs to

original_id = None
An external id for the person, e.g. a user id from Twitter

username = None
Username is a short system-y name.

full_name = None
Full name is a longer user-friendly name

language
The person's primary *Language*

message_count = None
The number of messages the person produced

replied_to_count = None
The number of times the person's messages were replied to

shared_count = None
The number of times the person's messages were shared or retweeted

mentioned_count = None
The number of times the person was mentioned in other people's messages

friend_count = None
The number of people this user has connected to

follower_count = None
The number of people who have connected to this person

profile_image_url = None
The person's profile image url

class msgvis.apps.corpus.models.**Message**(**args*, ***kwargs*)
The Message is the central data entity for the dataset.

dataset
Which *Dataset* the message belongs to

original_id = None
An external id for the message, e.g. a tweet id from Twitter

type
The *MessageType* Message type: retweet, reply, origin...

sender
The *Person* who sent the message

time = None
The datetime.datetime (in UTC) when the message was sent

language
The *Language* of the message.

sentiment = None
The sentiment label for message.

timezone
The *Timezone* of the message.

replied_to_count = None
The number of replies this message received.

shared_count = None
The number of times this message was shared or retweeted.

contains_hashtag = None
True if the message has a *Hashtag*.

contains_url = None
True if the message has a *Url*.

contains_media = None
True if the message has any *Media*.

contains_mention = None
True if the message mentions any *Person*.

urls
The set of *Url* in the message.

hashtags
The set of *Hashtag* in the message.

media
The set of *Media* in the message.

mentions
The set of *Person* mentioned in the message.

text = None
The actual text of the message.

1.6 Datatable

The Datatable app is responsible for generating and returning visualization data for specific configurations of dimensions and filters.

1.7 Questions

The Questions app is concerned with persisting research questions and articles to the database and retrieving research questions that correspond to current dimension selections.

1.7.1 Models

class msgvis.apps.questions.models.**Article**(**args*, ***kwargs*)
A published research article.

year = None
The publication year for the article.

authors = None
A plain-text author list.

link = None
A url to the article.

title = None
The title of the article.

venue = None
The venue where the article was published.

class msgvis.apps.questions.models.**Question**(**args*, ***kwargs*)
A research question from an *Article*. May be associated with a number of *DimensionKey* objects.

source
The source article for the question.

text = None
The text of the question.

dimensions
A set of dimensions related to the question.

classmethod get_sample_questions(**dimension_list*)
Given dimensions, return sample research questions.

class msgvis.apps.questions.models.**QuestionDimensionConnection**(*id*, *question_id*, *dimension_id*, *count*)

1.8 API

The purpose of the API is to provide access to statistical summaries of the message database that can be used to render visualizations. With many of the API requests, a JSON object should be provided that indicates the user's current interest and affects how the results will be delivered.

1.8.1 API Objects

This module defines serializers for the main API data objects:

DimensionSerializer	JSON representation of Dimensions for the API.
FilterSerializer	Filters indicate a subset of the range of a specific dimension.
MessageSerializer	JSON representation of *Message* objects for the API.
QuestionSerializer	JSON representation of a *Question* object for the API.

class msgvis.apps.api.serializers.**DimensionSerializer**(*instance=None*, *data=<class rest_framework.fields.empty>*, ***kwargs*)

JSON representation of Dimensions for the API.

Dimension objects describe the variables that users can select to visualize the dataset. An example is below:

```
{
  "key": "time",
  "name": "Time",
  "description": "The time the message was sent",
}
```

class msgvis.apps.api.serializers.**FilterSerializer**(*instance=None*, *data=<class rest_framework.fields.empty>*, ***kwargs*)

Filters indicate a subset of the range of a specific dimension. Below is an array of three filter objects.

```
[{
  "dimension": "time",
  "min_time": "2010-02-25T00:23:53Z",
  "max_time": "2010-02-28T00:23:53Z"
},
{
  "dimension": "words",
  "levels": [
    "cat",
    "dog",
    "alligator"
  ]
},
{
  "dimension": "reply_count",
  "max": 100
}]
```

Although every filter has a `dimension` field, the specific properties vary depending on the type of the dimension and the kind of filter.

At this time, there are three types of filters:

- Quantitative dimensions can be filtered using one or both of the `min` and `max` properties (inclusive).
- The time dimension can be filtered using one or both of the `min_time` and `max_time` properties (inclusive).
- Categorical dimensions can be filtered by specifying an `include` list. All other items are assumed to be excluded.

The 'value' field may also be used for exact matches.

class msgvis.apps.api.serializers.**MessageSerializer**(*instance=None*, *data=<class rest_framework.fields.empty>*, ***kwargs*)

JSON representation of *Message* objects for the API.

Messages are provided in a simple format that is useful for displaying examples:

```
{
  "id": 52,
  "dataset": 2,
  "text": "Some sort of thing or other",
```

```
    "sender": {
      "id": 2,
      "dataset": 1
      "original_id": 2568434,
      "username": "my_name",
      "full_name": "My Name"
    },
    "time": "2010-02-25T00:23:53Z"
}
```

Additional fields may be added later.

class msgvis.apps.api.serializers.**QuestionSerializer**(*instance=None*, *data=<class rest_framework.fields.empty>*, ***kwargs*)

JSON representation of a *Question* object for the API.

Research questions extracted from papers are given in the following format:

```
{
  "id": 5,
  "text": "What is your name?",
  "source": {
    "id": 13,
    "authors": "Thingummy & Bob",
    "link": "http://ijn.com/3453295",
    "title": "Names and such",
    "year": "2001",
    "venue": "International Journal of Names"
  },
  "dimensions": ["time", "author_name"]
}
```

The `source` object describes a research article reference where the question originated.

The `dimensions` list indicates which dimensions the research question is associated with.

1.8.2 API Endpoints

The view classes below define the API endpoints.

Endpoint	Url	Purpose
Get Data Table	/api/table	Get table of counts based on dimensions/filters
Get Example Messages	/api/messages	Get example messages for slice of data
Get Research Questions	/api/questions	Get RQs related to dimensions/filters
Message Context	/api/context	Get context for a message
Snapshots	/api/snapshots	Save a visualization snapshot

class msgvis.apps.api.views.**DataTableView**(***kwargs*)

Get a table of message counts or other statistics based on the current dimensions and filters.

The request should post a JSON object containing a list of one or two dimension ids and a list of filters. A `measure` may also be specified in the request, but the default measure is message count.

The response will be a JSON object that mimics the request body, but with a new `result` field added. The result field includes a `table`, which will be a list of objects.

Each object in the table field represents a cell in a table or a dot (for scatterplot-type results). For every dimension in the dimensions list (from the request), the result object will include a property keyed to the name of the

dimension and a value for that dimension. A `value` field provides the requested summary statistic.

The `result` field also includes a `domains` object, which defines the list of possible values within the selected data for each of the dimensions in the request.

This is the most general output format for results, but later we may switch to a more compact format.

Request: `POST /api/table`

Format: (request without `result` key)

```
{
  "dataset": 1,
  "dimensions": ["time"],
  "filters": [
    {
      "dimension": "time",
      "min_time": "2015-02-25T00:23:53Z",
      "max_time": "2015-02-28T00:23:53Z"
    }
  ],
  "result": {
    "table": [
      {
        "value": 35,
        "time": "2015-02-25T00:23:53Z"
      },
      {
        "value": 35,
        "time": "2015-02-26T00:23:53Z"
      },
      {
        "value": 35,
        "time": "2015-02-27T00:23:53Z"
      },
      {
        "value": 35,
        "time": "2015-02-28T00:23:53Z"
      },
      "domains": {
        "time": [
          "some_time_val",
          "some_time_val",
          "some_time_val",
          "some_time_val"
        ]
      ],
      "domain_labels": {}
}
```

class `msgvis.apps.api.views.`**`ExampleMessagesView`**(***kwargs*)

Get some example messages matching the current filters and a focus within the visualization.

Request: `POST /api/messages`

Format:: (request should not have `messages` key)

```
{
    "dataset": 1,
    "filters": [
        {
```

```
                "dimension": "time",
                "min_time": "2015-02-25T00:23:53Z",
                "max_time": "2015-02-28T00:23:53Z"
            }
        ],
        "focus": [
            {
                "dimension": "time",
                "value": "2015-02-28T00:23:53Z"
            }
        ],
        "messages": [
            {
                "id": 52,
                "dataset": 1,
                "text": "Some sort of thing or other",
                "sender": {
                    "id": 2,
                    "dataset": 1
                    "original_id": 2568434,
                    "username": "my_name",
                    "full_name": "My Name"
                },
                "time": "2015-02-25T00:23:53Z"
            }
        ]
    }
```

class msgvis.apps.api.views.**KeywordMessagesView**(***kwargs*)
Get some example messages matching the keyword.

Request: POST /api/search

Format:: (request should not have messages key)

```
    {
        "dataset": 1,
        "keywords": "soup ladies,food,NOT job",
        "messages": [
            {
                "id": 52,
                "dataset": 1,
                "text": "Some sort of thing or other",
                "sender": {
                    "id": 2,
                    "dataset": 1
                    "original_id": 2568434,
                    "username": "my_name",
                    "full_name": "My Name"
                },
                "time": "2015-02-25T00:23:53Z"
            }
        ]
    }
```

class msgvis.apps.api.views.**ActionHistoryView**(***kwargs*)
Add a action history record.

Request: POST /api/history

Format:: (request should not have `messages` key)

```
{
   "records": [
       {
            "type": "click-legend",
            "contents": "group 10"
        },
        {
            "type": "group:delete",
            "contents": "{\"group\": 10}"
        }
    ]
}
```

class msgvis.apps.api.views.**GroupView**(***kwargs*)
Get some example messages matching the keyword.

Request: `POST /api/group`

Format:: (request should not have `messages` key)

```
{
    "dataset": 1,
    "keyword": "like",
    "messages": [
        {
            "id": 52,
            "dataset": 1,
            "text": "Some sort of thing or other",
            "sender": {
                "id": 2,
                "dataset": 1
                "original_id": 2568434,
                "username": "my_name",
                "full_name": "My Name"
            },
            "time": "2015-02-25T00:23:53Z"
        }
    ]
}
```

class msgvis.apps.api.views.**KeywordView**(***kwargs*)
Get top 10 keyword results.

Request: `GET /api/keyword?dataset=1&q= [...]`

```
{
    "dataset": 1,
    "q": "mudslide oso",
    "keywords": ["mudslide oso", "mudslide oso soup", "mudslide oso ladies"]
}
```

class msgvis.apps.api.views.**ResearchQuestionsView**(***kwargs*)
Get a list of research questions related to a selection of dimensions and filters.

Request: `POST /api/questions`

Format: (request without `questions` key)

```
    {
        "dimensions": ["time", "hashtags"],
        "questions": [
            {
              "id": 5,
              "text": "What is your name?",
              "source": {
                "id": 13,
                "authors": "Thingummy & Bob",
                "link": "http://ijn.com/3453295",
                "title": "Names and such",
                "year": "2001",
                "venue": "International Journal of Names"
              },
              "dimensions": ["time", "author_name"]
            }
        ]
    }
```

class msgvis.apps.api.views.**DatasetView**(***kwargs*)
Get details of a dataset

Request: `GET /api/dataset/1`

class msgvis.apps.api.views.**APIRoot**(***kwargs*)
The Text Visualization DRG Root API View.

CHAPTER 2

Development Setup

To run this project, you can either set up your own machine or use a virtual Ubuntu machine with Vagrant. There are separate instructions for each below:

2.1 Run in a VM

There is configuration included to run this project inside an Ubuntu virtual machine controlled by Vagrant. This is especially recommended on Windows. If you go this route, you can skip the Manual Setup section below.

Instead, follow these steps:

1. Install Vagrant and Virtualbox
2. Start the virtual machine.

This will download a basic Ubuntu image, install some additional software on it, and perform the initial project setup.

Note: If you are on windows: You should run this command in an Administrator cmd.exe or Powershell.

```
$ vagrant up
```

If you are on mac: You need to make sure the setup script has executable permission.

```
chmod a+x setup/scripts/dev_setup.sh
```

---

3. Once your Ubuntu VM is started, you can SSH into it with `vagrant ssh`. This will use a key-based authentication to log you into the VM.

You can also log in using any SSH client (e.g. PuTTY), at `localhost:2222`. The username and password are both `vagrant`, or you can also configure key-based auth: use `vagrant ssh-config` to find the private key for accessing the VM.

When you log in, your terminal will automatically drop into a Python virtualenv and cd to `/home/vagrant/textvisdrg`.

2.2 Manual Setup

You will need to have the following packages installed:

- MySQL 5.5
- Python 2.7 and pip
- virtualenv
- virtualenvwrapper (recommended)
- Node.js
- Bower

Once you have the above prerequisites working, clone this repository to your machine.

Go to the directory where you have cloned the repository and run the setup script, as below:

```
$ cd textvisdrg
$ ./setup/scripts/dev_setup.sh
```

This script will perform the following steps for you:

1. Check that your system has the prerequisites available.
2. Prompt you for database settings. If it can't reach the database, it will give you a snippet of MySQL code needed to create the database with the supplied settings.
3. Create a Python virtual environment. This keeps Python packages needed for this project from interfering with any other packages you already have installed on your system.
4. Creates a `.env` file in your project directory that sets environment variables for Django, most importantly the database connection settings.
5. Installs python packages, NPM packages, and bower packages (using the `fab dependencies` command).
6. Runs the database migrations (using `fab migrate`).

CHAPTER 3

Workflow

This page explains how to develop this software and the various processes involved. For now, refer to the *fabfile* for useful shortcut commands.

3.1 Fabric Commands

Define common admin and maintenance tasks here. For more info: http://docs.fabfile.org/en/latest/

fabfile.**pip_install**(*environment='dev'*)
Install pip requirements for an environment: test, prod, [dev]

fabfile.**dependencies**(*default_env='dev'*)
Install requirements for pip, npm, and bower all at once.

fabfile.**test**(*settings_module='msgvis.settings.test'*)
Run tests

fabfile.**test_coverage**(*settings_module='msgvis.settings.test'*)
Run tests with coverage

fabfile.**make_test_data**(*outfile=path(u'/home/docs/checkouts/readthedocs.org/user_builds/textvisdrg/checkouts/latest/setup/fi*
Updates the test_data.json file based on what is in the database

fabfile.**load_test_data**(*infile=path(u'/home/docs/checkouts/readthedocs.org/user_builds/textvisdrg/checkouts/latest/setup/fix*
Load test data from test_data.json

fabfile.**generate_fixtures**()
Regenerate configured fixtures from the database.

fabfile.**load_fixtures**()
Replaces the database tables with the contents of fixtures.

fabfile.**import_corpus**(*dataset_file_or_dir*)
Import a dataset from a file

fabfile.**restart_webserver**()
Restart a local gunicorn process

fabfile.**supervisor**()
Starts the supervisor process

fabfile.**deploy**(*branch=None*)
SSH into a remote server, run commands to update deployment, and start the server.

This requires that the server is already running a fairly recent copy of the code.

Furthermore, the app must use a

fabfile.**topic_pipeline**(*dataset*, *name='my topic model'*, *num_topics=30*)
Run the topic pipeline on a dataset

fabfile.**info**()
Print a bunch of info about the environment

fabfile.**nltk_init**()
Download required nltk corpora

fabfile.**memcached_status**()
Display the status of the memcached server

3.1.1 Extra Fabric Commands

A collection of runnable fabric tasks. Make sure to call conf.configure() first!

fabutils.tasks.**pull**()
Just runs git pull

fabutils.tasks.**manage**(*command*)
Run a Django management command.

fabutils.tasks.**migrate**()
Runs migrations

fabutils.tasks.**build_static**()
Builds static files for production

fabutils.tasks.**docs**(*easy=None*)
Build the documentation

fabutils.tasks.**runserver**()
Runs the Django development server

fabutils.tasks.**reset_db**()
Removes all of the tables

fabutils.tasks.**clear_cache**()
Deletes the cached static files

fabutils.tasks.**interpolate_env**(*outpath=None*)
Writes a .env file with variables interpolated from the current environment

fabutils.tasks.**check_database**()
Makes sure the database is accessible

fabutils.tasks.**print_env**()
Print the local .env file contents

fabutils.tasks.**npm_install**()
Install npm requirements

fabutils.tasks.**bower_install**()
Install bower requirements

CHAPTER 4

Indices and tables

- genindex
- modindex

f

m

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

Y