**Rasa setup**:

* Download & Install Python: <https://www.python.org/downloads/>
* Follow instructions here: <https://rasa.com/docs/getting-started/>
* running **rasa init** in your current directory sets up a full-blown project in your current directory
* alternately, you can copy the contents of an existing project (e.g. <https://github.com/ryanmark1867/chatbot>):
  + in a new directory (root directory for the project):
    - copy the following files from the existing project in your new directory (root directory for the project:
      * actions.py (contains classes that define the custom actions taken for intents defined in the domain.yml file)
      * domain.yml (defines intents, entities, slots, utterances etc)
      * config.yml (defines rasa pipeline)
      * credentials.yml
      * endpoints.yml (defines local server that serves Python functions defined in actions.py file)
    - create subdirectory “data” and copy the following files into it:
      * nlu.md (training data)
      * stories.md files (defines flow of intents from user and actions from the Rasa system

**Running a Rasa session**:

* if you change any of the .yml or .md files listed in the Rasa setup session, you need to retrain the Rasa model in order for these changes to take effect:
  + Run “**rasa train**” in the root directory of the project
* if you change any of the custom actions defined in classes in the actions.py file, you need to run the following command for these changes to take effect:
  + Run “**rasa run actions**” in the root directory of the project
* To exercise rasa in the GUI interface Rasa X:
  + Run “**rasa x**” in the root directory
  + Once Rasa x starts, click on the model tab and ensure that the model you want to exercise is **Active** by clicking on the three dots at the right for the model you want. If you want to select a brand new model to be active, you must first make another model active and then the three dots will appear for the new model and allow you to make it active

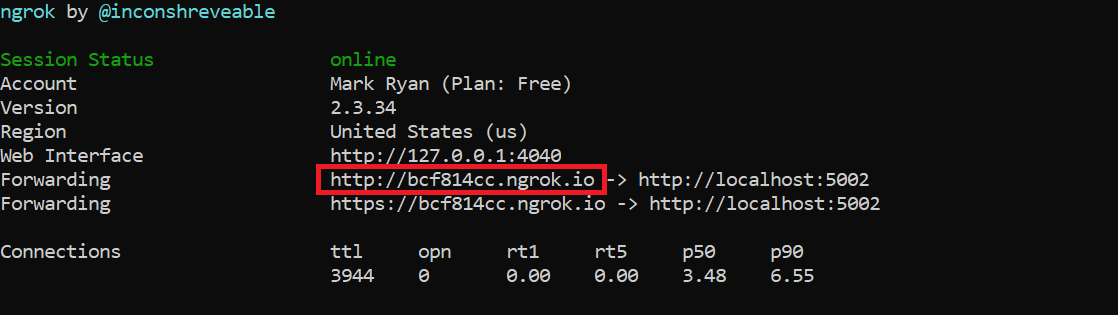
**Interactively training a Rasa model in Rasa x**:

* In Rasa X you have two modes to interact with the model
  + Default: you as user enter responses and model responds. This mode does not result in any additional training for the model
  + Interactive (select in the top right on the first tab in Rasa X). In this mode you validate how the model interprets every interaction. If you agree with the next step the model comes up with, click on the purple checkmark in the bottom left. If you don’t agree, you can select a different step and click on the checkmark to record your change. NOTE: it is critical that you backup the nlu.md file prior to using this mode. This mode updates this file with no warning, so you can end up losing work if you don’t backup the nlu.md file before going into this mode.

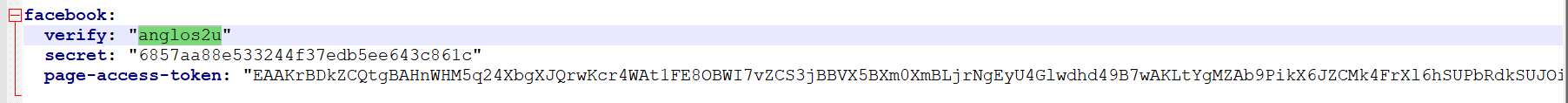
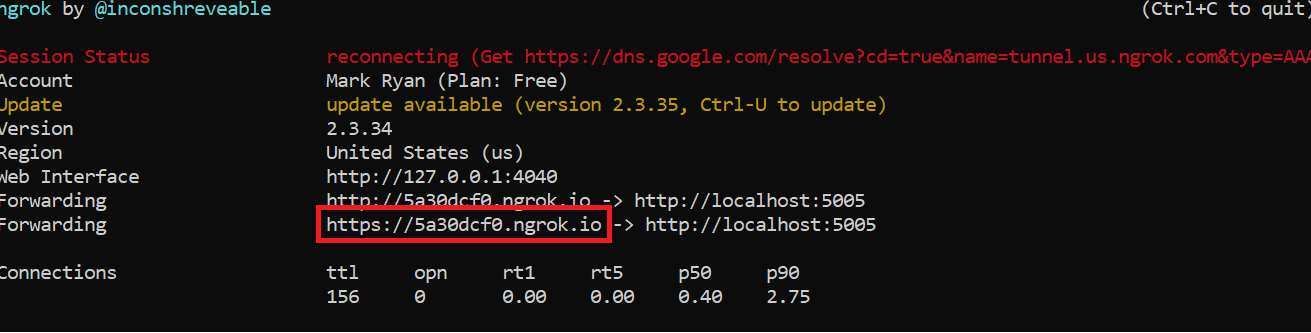
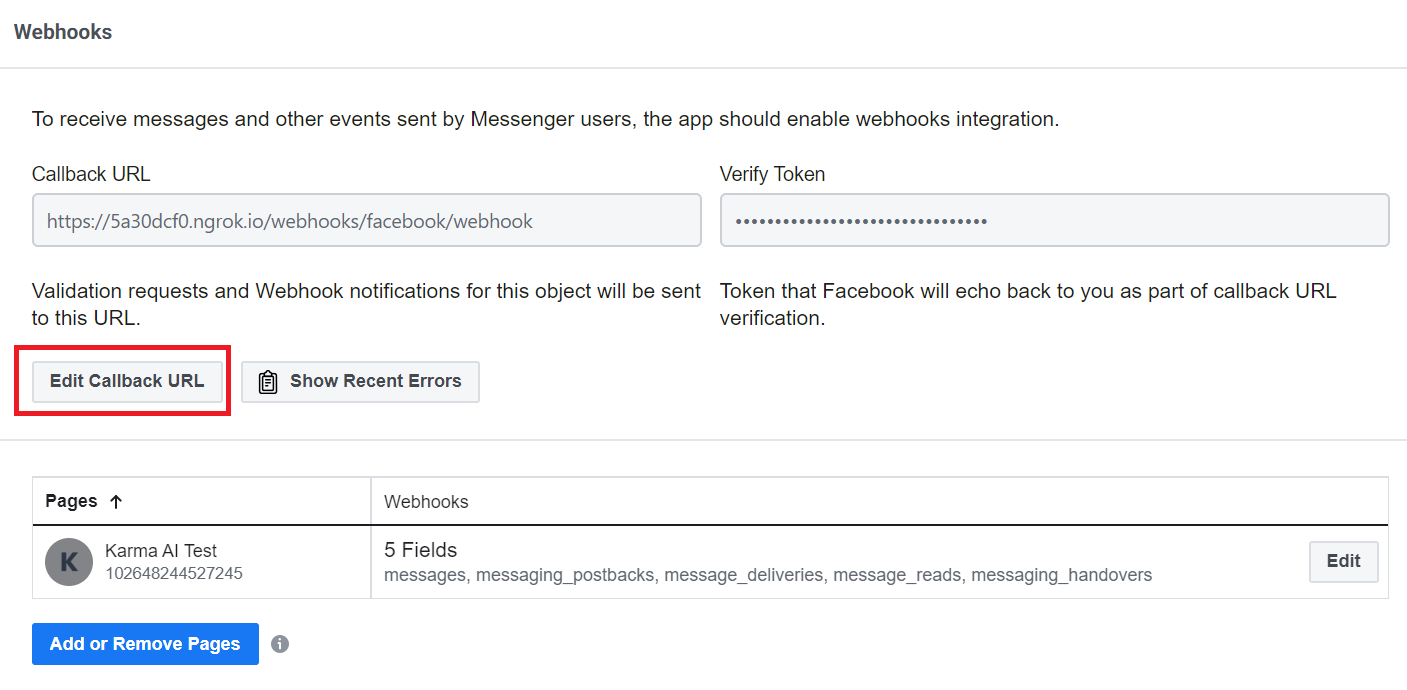
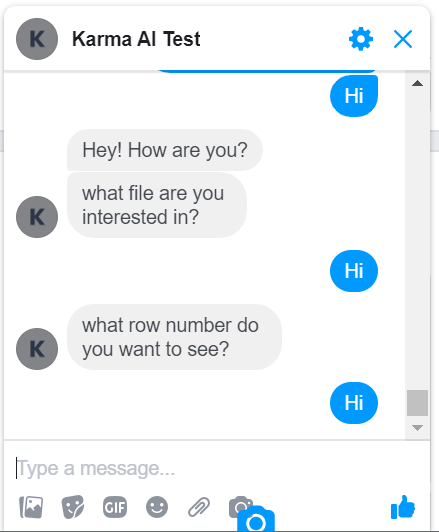
**Specifying actions in actions.py file**:

* The Rasa documentation lists several options for defining custom actions (code that gets executed in response to user input), but the simplest approach is to define the custom actions in the actions.py file.
* Rasa setup includes action server for Python that sets everything up for you in terms of serving the Python code – “all” you have to do is update the code in the actions.py file to have a class corresponding with each custom action you define in the domain.yml file
* Gotchas for the actions.py file:
  + Documentation for passing information between the Rasa layer and the Python code in actions.py is particularly horrible
  + You pass parameters to the Python classes using Slots defined in the domain.yml file and given values in the live interactions
  + In actions.py, the tracker object contains the values of all Slots. Any Slots that have not had their values set at the Rasa layer have a value of “None”
  + From actions.py you can deliver text directly to the Rasa chat interaction using dispatcher.utter\_message
    - dispatcher.utter\_message("Here is the list of column names for file:")
  + **IMPORTANT**: the argument for dispatcher.utter\_message \*must\* be a single string. Anything else an you will get an arcane error message in the command window where you invoked “run rasa actions”. This means:
    - If you want to send any messages back to the chatbot session that are made up of text plus a slot value or some other derived value, you first have to define a single string that concatenates everything, and then pass that string as an argument to dispatcher.utter\_message
    - If you want to send a non-string value (like an integer or a float), you must first cast it to a string

**Serving Rasa X running on a local machine using ngrok:**

* The above setup instructions are for running Rasa entirely on a local machine
* If you want to tactically server the local instance of Rasa X so that others can exercise the model, you can use ngrok – see discussion starting at 45:00 here <https://www.youtube.com/watch?v=YCZsiyOfDLk&t=496s>
* Ngrok saga:
  + Setup ngrok - <https://ngrok.com/download>
  + In the directory where you setup ngrok, run:
    - .\Ngrok http 5002
    - You will get output like this – copy the Forwarding URL (highlighted in the red box below)
    - 
    - On linux, EXPORT RASA\_X\_HOSTNAME=*forwarding URL from above*
    - On Windows, define the following env variables:
      * RASA\_X\_HOSTNAME with value of ngrok forwarding URL
      * RASA\_X\_HOST with value of ngrok forwarding URL
      * RASA\_X\_PASSWORD with your choice of password – you will need this when you start up Rasa X
    - “rasa x” in the root directory of your project to start Rasa X
    - Windows:
      * once Rasa X starts, replace “localhost:5002” in the URL with the ngrok forwarding URL and refresh
      * If prompted, enter password defined above
    - Once you have the Rasa X interface going, click on the share icon in the upper right and generate shareable URL – you can share the resulting URL with others who are then able to exercise the model on their own systems
    - NOTE: response may be slow – users may have to wait a minute or more for response text to get shown in their Rasa session

**Running rasa chatbot via Facebook Messenger (FM)**

* Starting docs – none of which is 100% reliable:
  1. <https://rasa.com/docs/rasa/user-guide/connectors/facebook-messenger/> - missing details of how to set up webhook in Facebook
  2. <https://medium.com/@tatiana.parshina/connecting-rasa-ai-chatbot-to-facebook-messenger-6d024e642dbd> - more details, but still missing essential details (such as what token to use in webhook), missing steps in FM setup, and misleading links to FM scripts that aren’t needed to get it working
* Key points:
  + Don’t use rasa-x; just FM by itself
  + Token required by webhook definition in Facebook is the token in the “verified” field of the credentials.yml file in Rasa, **not** the page access token provided by FB following steps in 1 above:
  + 
* Here’s a summary end-to-end:
  + Start rasa:
    - rasa run --credentials credentials.yml
  + Start ngrok:
    - Go to ngrok directory and run:
      * .\Ngrok http 5005
    - Get the resulting https URL:
      * 
    - Webhook URL will be <ngrokurl>/webhooks/facebook/webhook
  + Set webhook callback in FB (using URL built in previous step <https://developers.facebook.com/apps/750984582087384/messenger/settings/>
  + 
  + **Now you should be able to message Karma AI Test and interact with the rasa bot:**
  + ****
  + Still working on an efficient way to reboot bot and to display images in FM

**Other gotchas**:

* Rasa documentation <https://rasa.com/docs/rasa/> is really, really terrible:
  + Many critical syntax issues are glossed over
  + Examples are either cursory or incomplete
* Join rasa forum <https://forum.rasa.com/> but beware. While responses from the community are reasonably speedy, they tend to be unhelpful. This is a problem because many aspects of the function lack examples and documentation
* It is really easy to screw up the syntax in the .yml files. The only way I have found to avoid introducing errors is to make one or two changes at a time and then run “rasa train” to validate the changes. This slows down progress, but attempting to make more than a handful of changes before retraining causes breakage with useless error messages
* Windows paths (with backslashes) get screwed up – backslashes get removes / duplicated. I move the data to files in git because of this problem
* To get rid of objects (slots, intents, etc) that are causing problems, you need to really clean house:
  + Create a fresh root directory and copy in the 6 main files (nlu.md, stories.md, domain.yml, config.yml, endpoints.yml, actions.py)
  + Ensure all processes related to rasa h