**Project Overview**

**Overview**

In this project, you will make use of Python to explore data related to bikeshare systems for three major bikeshare systems in the United States. You will perform data wrangling to unify the format of data from the three systems and write code to compute descriptive statistics. You will also make use of a package that is not part of the standard Python library to help you visualize the data.

If you are not familiar with the data analysis process, this project will be your first exposure to the kinds of steps that a data analyst takes when they approach a dataset. The steps of the process that you go through here will be covered in more depth as you continue on with the Nanodegree program. For now, all you need is the general programming skills you've learned from the "Introduction to Python Programming" course and a desire to learn about the data analysis process! Note that you may not have seen some of the packages used in the project prior to this point, so you should be comfortable with following documentation links and experimenting with code to build your intuition for how to use the packages to solve the problems posed.

# Project Details

# Getting Started

To complete this project, you will use Jupyter Notebooks through a workspace in the classroom. Jupyter Notebooks are a great way to work with your code interactively while also being able to include descriptive and informative text to build a report. The next few concepts in this lesson will help you get started with understanding notebooks. If you would like to learn more about these tools, or you need some additional help to get started, you can check out the course on Anaconda and Jupyter Notebooks [**here**](https://classroom.udacity.com/nanodegrees/nd002-mena/parts/8a8eea10-0c4c-4969-98db-3a2f8c651da6). You will also experience these lessons in the "Intro to Data Analysis" course, so you will learn this content later if you don't want to visit those pages now.

# Completing This Project

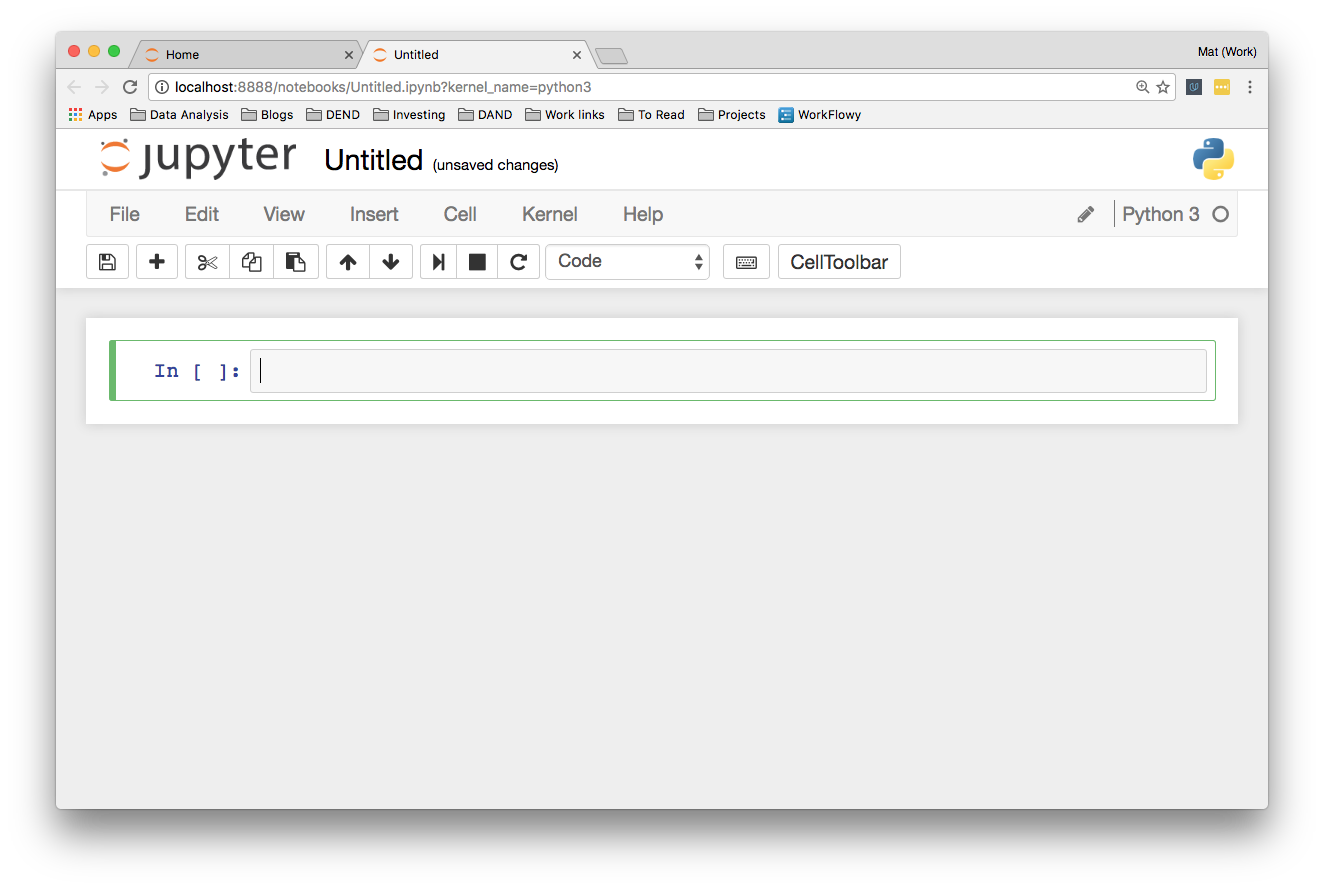
The Jupyter Notebook in the "Project Notebook" concept has all the information required to submit this project. Click on the Bike\_Share\_Analysis.ipynb file to start up the notebook kernel. Read through the notebook, answering questions and running blocks of code when prompted. Remember that if you get stuck or if you have any questions along the way, you can reach out for help using support resources like live help and the Slack community. If you feel comfortable with it, you can also download all of the necessary files, either from the Resources tab on this page or from the "Project Notebook" page, and work on the project using a local installation of Anaconda and Jupyter Notebook.

When you’ve worked through the entire notebook, you will download your work and submit it in the last concept in the lesson. You will submit two files: the original file and a more portable format. For the former, if you go to the front menu of the notebook page (where you can see the project files) and click on the checkbox to the left of the file name, you should see a set of buttons appear at the top of the page. You can click on the "Download" button to download your completed notebook to your local computer. For the latter, you can save a copy of the report as a .html or .pdf document by accessing the "File > Download as..." menu in the Notebook app. This way, someone who wants to just read your completed work can view it without needing to start up the Notebook app themselves.

# Notebook interface

# Notebook interface

When you create a new notebook, you should see something like this:

**[[](https://classroom.udacity.com/nanodegrees/nd002-ent/parts/8ba0d7fa-a87e-431a-9ff0-1e6090b7a6e5/modules/c89c8727-97d9-41ee-b57b-e7add630381a/lessons/ef631031-1887-4bd0-8e85-84ba793885f8/concepts/0d3b93d8-fd93-4dde-92f2-55594d6d458e)](https://classroom.udacity.com/nanodegrees/nd002-ent/parts/8ba0d7fa-a87e-431a-9ff0-1e6090b7a6e5/modules/c89c8727-97d9-41ee-b57b-e7add630381a/lessons/ef631031-1887-4bd0-8e85-84ba793885f8/concepts/0d3b93d8-fd93-4dde-92f2-55594d6d458e)**

Feel free to try this yourself and poke around a bit.

You’ll see a little box outlined in green. This is called a cell. Cells are where you write and run your code. You can also change it to render Markdown, a popular formatting syntax for writing web content. I'll cover Markdown in more detail later. In the toolbar, click “Code” to change it to Markdown and back. The little play button runs the cell, and the up and down arrows move cells up and down.

When you run a code cell, the output is displayed below the cell. The cell also gets numbered, you see In [1]: on the left. This lets you know the code was run and the order if you run multiple cells. Running the cell in Markdown mode renders the Markdown as text.

## The tool bar

Elsewhere on the tool bar, starting from the left:

* The anachronistic symbol for "save," the floppy disk. Saves the notebook!
* The + button creates a new cell
* Then, buttons to cut, copy, and paste cells.
* Run, stop, restart the kernel
* Cell type: code, Markdown, raw text, and header
* Command palette (see next)
* Cell toolbar, gives various options for cells such as using them as slides

### Command palette

The little keyboard is the command palette. This will bring up a panel with a search bar where you can search for various commands. This is really helpful for speeding up your workflow as you don't need to search around in the menus with your mouse. Just open the command palette and type in what you want to do. For instance, if you want to merge two cells:

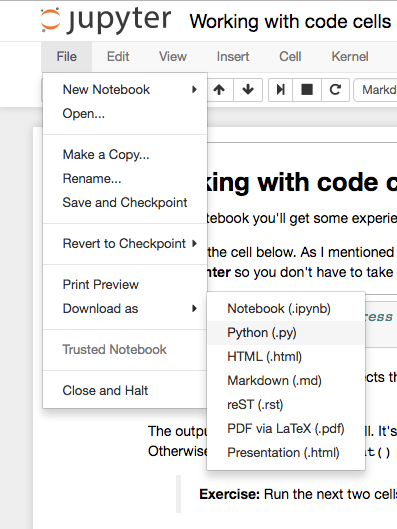
## More things

At the top you see the title. Click on this to rename the notebook.

Over on the right is the kernel type (Python 3 in my case) and next to it, a little circle. When the kernel is running a cell, it'll fill in. For most operations which run quickly, it won't fill in. It's a little indicator to let you know longer running code is actually running.

Along with the save button in the toolbar, notebooks are automatically saved periodically. The most recent save is noted to the right of the title. You can save manually with the save button, or by pressing escape then s on your keyboard. The escape key changes to command mode and s is the shortcut for "save." I'll cover command mode and keyboard shortcuts later.

In the "File" menu, you can download the notebook in multiple formats. You'll often want to download it as an HTML file to share with others who aren't using Jupyter. Also, you can download the notebook as a normal Python file where all the code will run like normal. The [**Markdown**](https://daringfireball.net/projects/markdown/) and **[reST](http://docutils.sourceforge.net/rst.html" \t "_blank)** formats are great for using notebooks in blogs or documentation.

**[[](https://classroom.udacity.com/nanodegrees/nd002-ent/parts/8ba0d7fa-a87e-431a-9ff0-1e6090b7a6e5/modules/c89c8727-97d9-41ee-b57b-e7add630381a/lessons/ef631031-1887-4bd0-8e85-84ba793885f8/concepts/0d3b93d8-fd93-4dde-92f2-55594d6d458e)](https://classroom.udacity.com/nanodegrees/nd002-ent/parts/8ba0d7fa-a87e-431a-9ff0-1e6090b7a6e5/modules/c89c8727-97d9-41ee-b57b-e7add630381a/lessons/ef631031-1887-4bd0-8e85-84ba793885f8/concepts/0d3b93d8-fd93-4dde-92f2-55594d6d458e)**

# Code Cells

# Code cells

Most of your work in notebooks will be done in code cells. This is where you write your code and it gets executed. In code cells you can write any code, assigning variables, defining functions and classes, importing packages, and more. Any code executed in one cell is available in all other cells. To give you some practice, I created a notebook below you can work through.

**MENU**

**EXPAND**

# Markdown cells

# Markdown cells

As mentioned before, cells can also be used for text written in Markdown. Markdown is a formatting syntax that allows you to include links, style text as bold or italicized, and format code. As with code cells, you press **Shift + Enter** or **Control + Enter** to run the Markdown cell, where it will render the Markdown to formatted text. Including text allows you to write a narrative alongside your code, as well as documenting your code and the thoughts that went into it.

You can find the [**documentation here**](https://daringfireball.net/projects/markdown/basics), but I'll provide a short primer.

## Headers

You can write headers using the pound/hash/[**octothorpe**](http://www.worldwidewords.org/weirdwords/ww-oct1.htm) symbol # placed before the text. One # renders as an h1 header, two #s is an h2, and so on. Looks like this:

# Header 1

## Header 2

### Header 3

renders as

# Header 1

## Header 2

### Header 3

## Links

Linking in Markdown is done by enclosing text in square brackets and the URL in parentheses, like this [Udacity's home page](https://www.udacity.com) for a link to [**Udacity's home page**](https://www.udacity.com/).

## Emphasis

You can add emphasis through bold or italics with asterisks or underscores (\* or \_). For italics, wrap the text in one asterisk or underscore, \_gelato\_ or \*gelato\*renders as gelato.

Bold text uses two symbols, \*\*aardvark\*\* or \_\_aardvark\_\_ looks like **aardvark**.

Either asterisks or underscores are fine as long as you use the same symbol on both sides of the text.

## Code

There are two different ways to display code, inline with text and as a code block separated from the text. To format inline code, wrap the text in backticks. For example, `string.punctuation` renders as string.punctuation.

To create a code block, start a new line and wrap the text in three backticks

```

**import** requests

response = requests.get('https://www.udacity.com')

```

or indent each line of the code block with four spaces.

**import** requests

response = requests.get('https://www.udacity.com')

## Math expressions

You can create math expressions in Markdown cells using **[LaTeX](https://www.latex-project.org/" \t "_blank)** symbols. Notebooks use MathJax to render the LaTeX symbols as math symbols. To start math mode, wrap the LaTeX in dollar signs $y = mx + b$ for inline math. For a math block, use double dollar signs,

$$

y = \frac{a}{b+c}

$$

This is a really useful feature, so if you don't have experience with LaTeX [**please read this primer**](http://data-blog.udacity.com/posts/2016/10/latex-primer/) on using it to create math expressions.

<https://s3.amazonaws.com/content.udacity-data.com/courses/ud1111/Markdown+cells.mp4>

## Wrapping up

Here's [**a cheatsheet**](https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet) you can use as a reference for writing Markdown. My advice is to make use of the Markdown cells. Your notebooks will be much more readable compared to a bunch of code blocks.

# Magic keywords

# Magic keywords

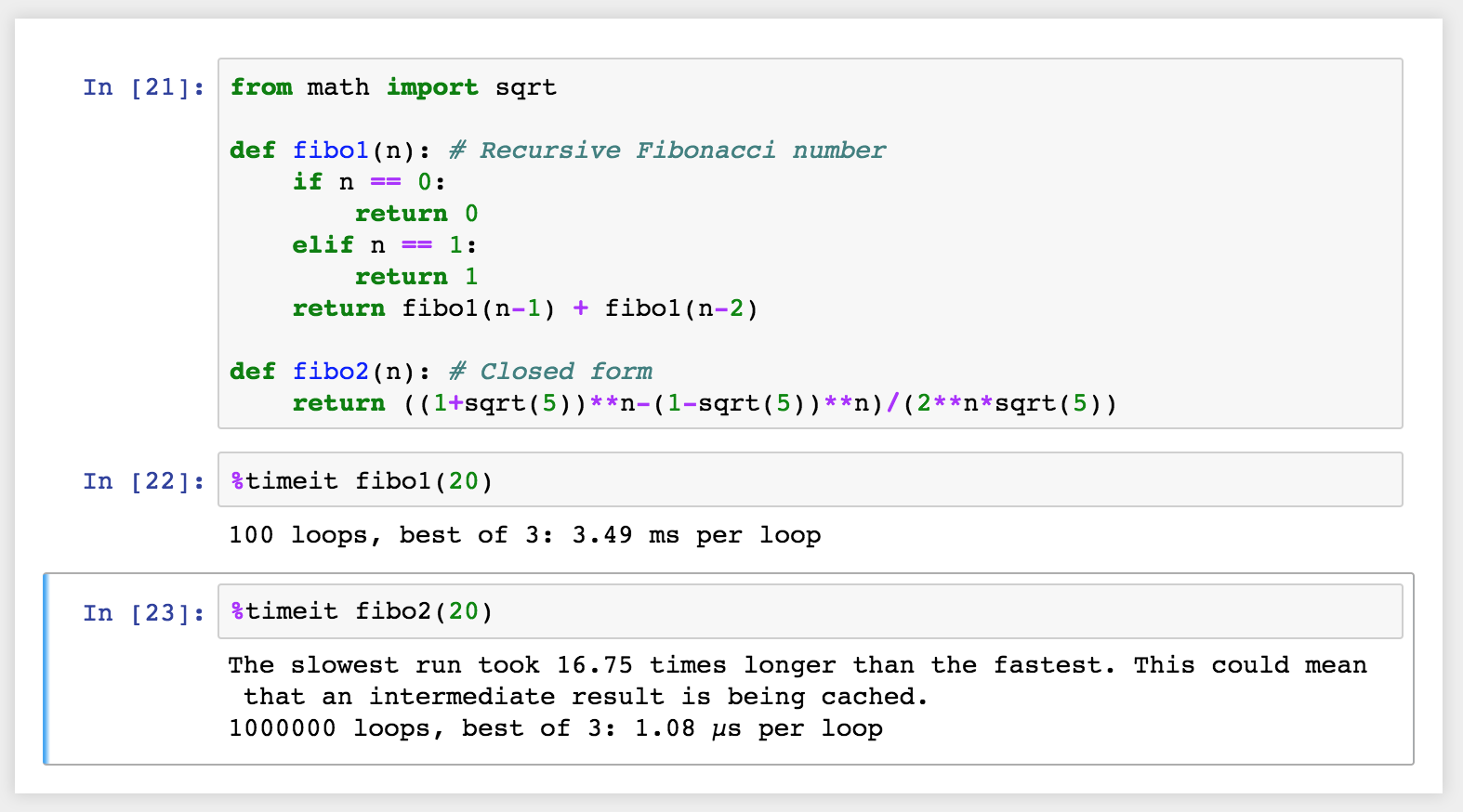
Magic keywords are special commands you can run in cells that let you control the notebook itself or perform system calls such as changing directories. For example, you can set up matplotlib to work interactively in the notebook with %matplotlib.

Magic commands are preceded with one or two percent signs (% or %%) for line magics and cell magics, respectively. Line magics apply only to the line the magic command is written on, while cell magics apply to the whole cell.

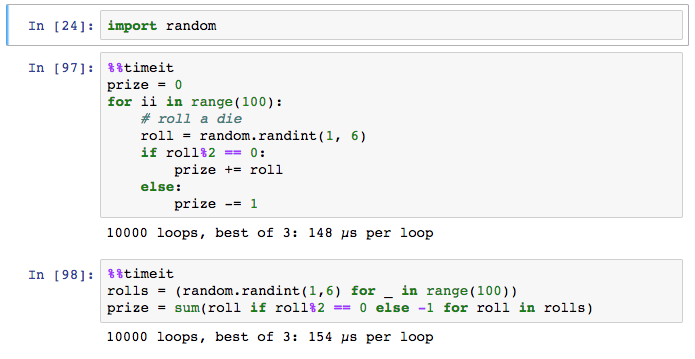
**NOTE:** These magic keywords are specific to the normal Python kernel. If you are using other kernels, these most likely won't work.

## Timing code

At some point, you'll probably spend some effort optimizing code to run faster. Timing how quickly your code runs is essential for this optimization. You can use the timeit magic command to time how long it takes for a function to run, like so:

**[[](https://classroom.udacity.com/nanodegrees/nd002-ent/parts/8ba0d7fa-a87e-431a-9ff0-1e6090b7a6e5/modules/c89c8727-97d9-41ee-b57b-e7add630381a/lessons/ef631031-1887-4bd0-8e85-84ba793885f8/concepts/256cdd36-17d4-442a-a033-7c64ce83f7f8)](https://classroom.udacity.com/nanodegrees/nd002-ent/parts/8ba0d7fa-a87e-431a-9ff0-1e6090b7a6e5/modules/c89c8727-97d9-41ee-b57b-e7add630381a/lessons/ef631031-1887-4bd0-8e85-84ba793885f8/concepts/256cdd36-17d4-442a-a033-7c64ce83f7f8)**

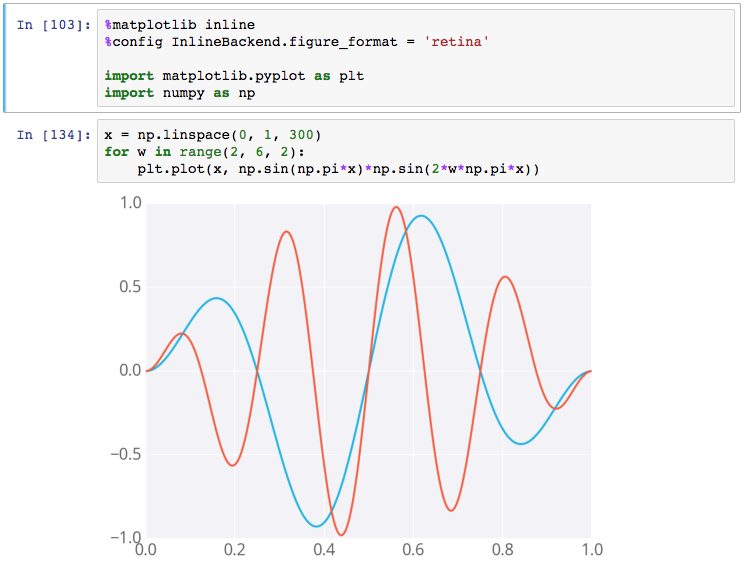
If you want to time how long it takes for a whole cell to run, you’d use %%timeit like so:

**[[](https://classroom.udacity.com/nanodegrees/nd002-ent/parts/8ba0d7fa-a87e-431a-9ff0-1e6090b7a6e5/modules/c89c8727-97d9-41ee-b57b-e7add630381a/lessons/ef631031-1887-4bd0-8e85-84ba793885f8/concepts/256cdd36-17d4-442a-a033-7c64ce83f7f8)](https://classroom.udacity.com/nanodegrees/nd002-ent/parts/8ba0d7fa-a87e-431a-9ff0-1e6090b7a6e5/modules/c89c8727-97d9-41ee-b57b-e7add630381a/lessons/ef631031-1887-4bd0-8e85-84ba793885f8/concepts/256cdd36-17d4-442a-a033-7c64ce83f7f8)**

## Embedding visualizations in notebooks

As mentioned before, notebooks let you embed images along with text and code. This is most useful when you’re using matplotlib or other plotting packages to create visualizations. You can use %matplotlib to set up matplotlib for interactive use in the notebook. By default figures will render in their own window. However, you can pass arguments to the command to select a specific [**"backend"**](http://matplotlib.org/faq/usage_faq.html#what-is-a-backend), the software that renders the image. To render figures directly in the notebook, you should use the inline backend with the command %matplotlib inline.

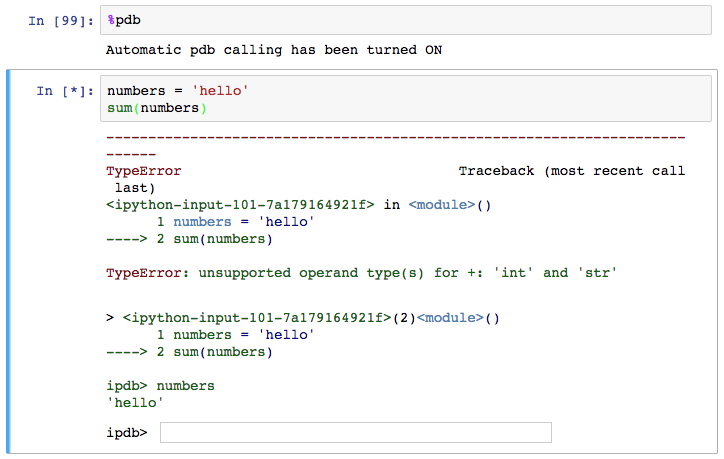
**Tip:** On higher resolution screens such as Retina displays, the default images in notebooks can look blurry. Use %config InlineBackend.figure\_format = 'retina' after %matplotlib inline to render higher resolution images.

**[[](https://classroom.udacity.com/nanodegrees/nd002-ent/parts/8ba0d7fa-a87e-431a-9ff0-1e6090b7a6e5/modules/c89c8727-97d9-41ee-b57b-e7add630381a/lessons/ef631031-1887-4bd0-8e85-84ba793885f8/concepts/256cdd36-17d4-442a-a033-7c64ce83f7f8)](https://classroom.udacity.com/nanodegrees/nd002-ent/parts/8ba0d7fa-a87e-431a-9ff0-1e6090b7a6e5/modules/c89c8727-97d9-41ee-b57b-e7add630381a/lessons/ef631031-1887-4bd0-8e85-84ba793885f8/concepts/256cdd36-17d4-442a-a033-7c64ce83f7f8)**

**[Example figure in a notebook](https://classroom.udacity.com/nanodegrees/nd002-ent/parts/8ba0d7fa-a87e-431a-9ff0-1e6090b7a6e5/modules/c89c8727-97d9-41ee-b57b-e7add630381a/lessons/ef631031-1887-4bd0-8e85-84ba793885f8/concepts/256cdd36-17d4-442a-a033-7c64ce83f7f8)**

## Debugging in the Notebook

With the Python kernel, you can turn on the interactive debugger using the magic command %pdb. When you cause an error, you'll be able to inspect the variables in the current namespace.

**[[](https://classroom.udacity.com/nanodegrees/nd002-ent/parts/8ba0d7fa-a87e-431a-9ff0-1e6090b7a6e5/modules/c89c8727-97d9-41ee-b57b-e7add630381a/lessons/ef631031-1887-4bd0-8e85-84ba793885f8/concepts/256cdd36-17d4-442a-a033-7c64ce83f7f8)](https://classroom.udacity.com/nanodegrees/nd002-ent/parts/8ba0d7fa-a87e-431a-9ff0-1e6090b7a6e5/modules/c89c8727-97d9-41ee-b57b-e7add630381a/lessons/ef631031-1887-4bd0-8e85-84ba793885f8/concepts/256cdd36-17d4-442a-a033-7c64ce83f7f8)**

**[Debugging in a notebook](https://classroom.udacity.com/nanodegrees/nd002-ent/parts/8ba0d7fa-a87e-431a-9ff0-1e6090b7a6e5/modules/c89c8727-97d9-41ee-b57b-e7add630381a/lessons/ef631031-1887-4bd0-8e85-84ba793885f8/concepts/256cdd36-17d4-442a-a033-7c64ce83f7f8)**

Above you can see I tried to sum up a string which gives an error. The debugger raises the error and provides a prompt for inspecting your code.

Read more about pdb in [**the documentation**](https://docs.python.org/3/library/pdb.html). To quit the debugger, simply enter q in the prompt.

## More reading

There are a whole bunch of other magic commands, I just touched on a few of the ones you'll use the most often. To learn more about them, [**here's the list**](http://ipython.readthedocs.io/en/stable/interactive/magics.html) of all available magic commands.

<https://github.com/Alicelibinguo/Python-Explore-BikeShare-Data/blob/master/Bike_Share_Analysis%20.py>