Week 12

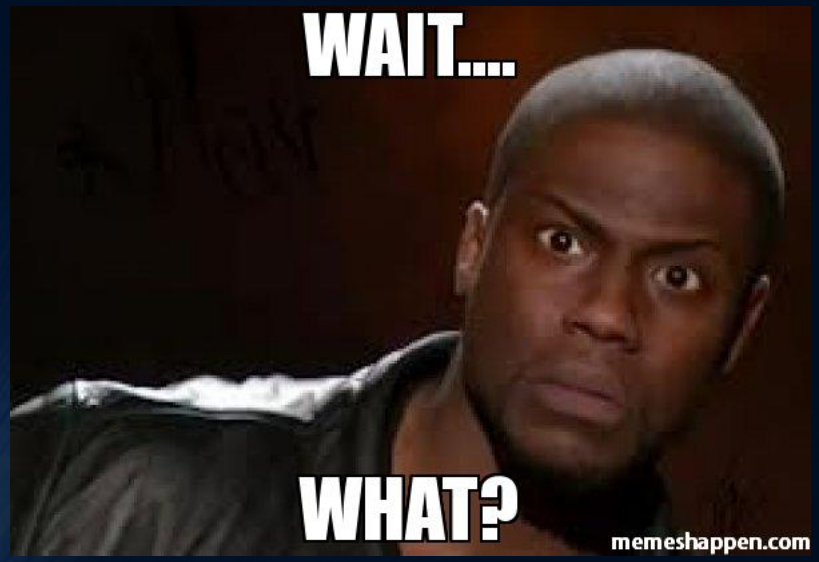
Key Concepts

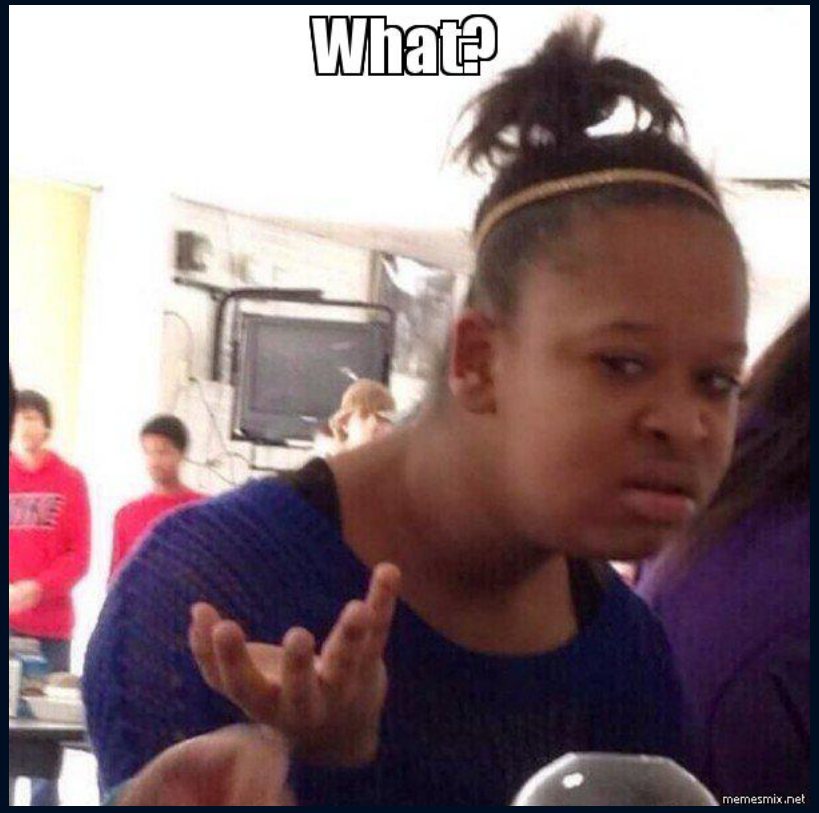
- Anaconda
  - What is Anaconda?
  - What is the Anaconda Distribution?
- What is a package manager?
  - Unix/Linux package managers
    - apt-get, yum, brew
  - Python package managers
    - pip & conda
- Virtual Environments
  - Why use a new one for each project?
- Packaging your Code

Anaconda

- What is Anaconda?
  - https://www.anaconda.com/what-is-anaconda/
  - Anaconda is the most popular data science *platform*
- Anaconda Distribution (Python & R)
  - Over 6 million users
  - Data science standard
  - Supports Windows, Mac OS, & Linux
  - Anaconda-curated packages
  - Dependency/package management

Wait...What even is a Python Distribution?





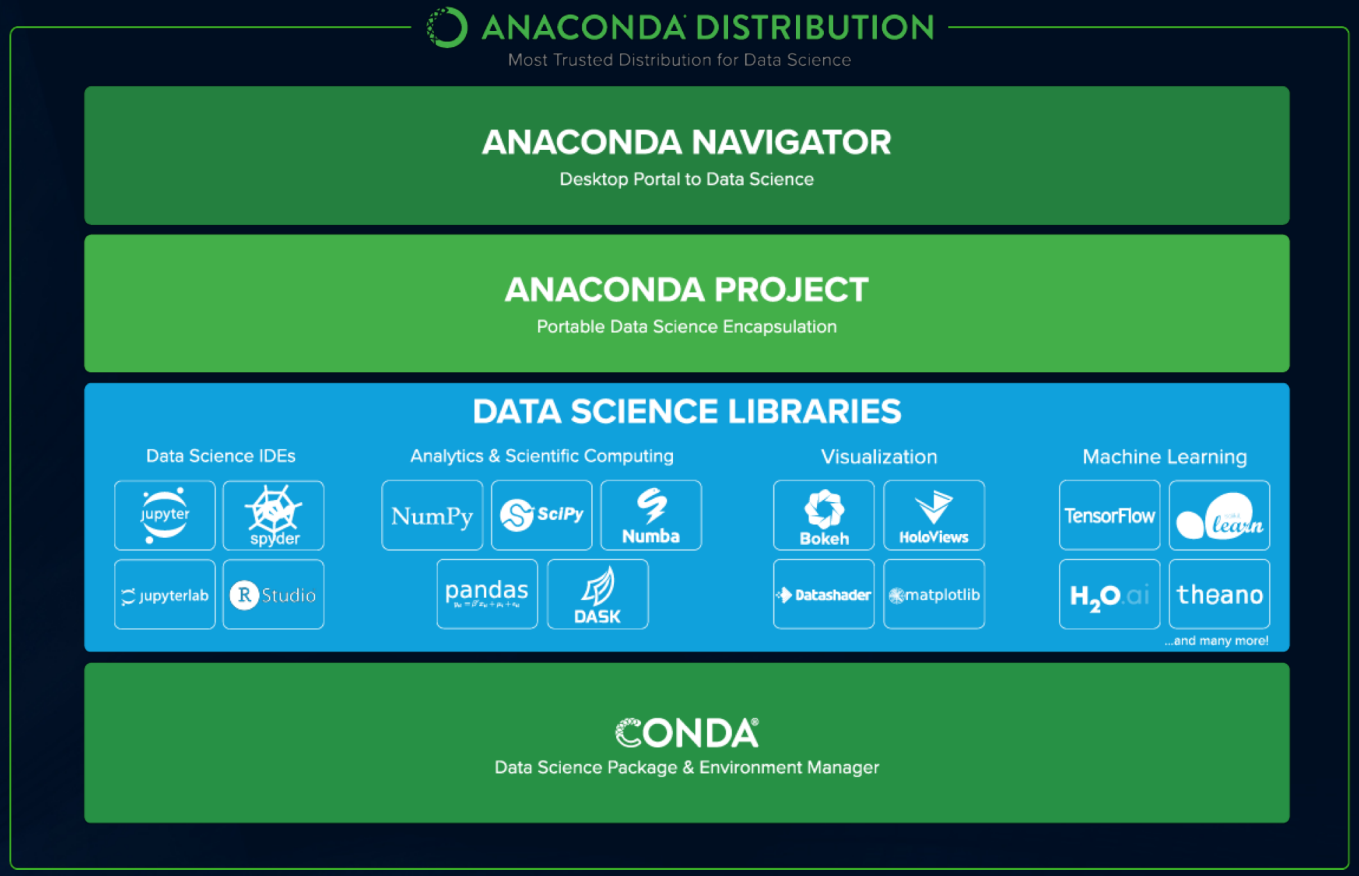
Distribution

- A ***distribution*** or ***distro*** is a collection of software components built, assembled, and configured so that it can be used essentially "as is" for its intended purpose
- Linux distros
  - Linux is very different from Windows and Mac OS in that it isn't produced by a single organization
    - Windows and Apple releasethe only new versions of their operating system
  - Different organizations and people work on different parts. There's the Linux kernel (the core of the operating system), the GNU shell utilities (the terminal interface and many of the commands you use), the X server (which produces a graphical desktop), the desktop environment (which runs on the X server to provide a graphical desktop), and more.
  - https://en.wikipedia.org/wiki/Linux_distribution#Types_and_trends

Python Distributions

- There are many different distributions of Python
- Each of these distributions is optimized for a specific purpose and, as a result, each has its pros and cons
- The *standard* distribution is called Cpython
  - Most broadly compatible across platforms, but has no accelerated math libraries
- Anaconda Python
  - Bundles most common libraries for commercial and scientific Python work
- https://www.infoworld.com/article/3267976/python/anaconda-cpython-pypy-and-more-know-your-python-distributions.html

Anaconda Distribution



What is a package manager?

- A **package manager** or **package management system** is a collection of software tools that automates the process of installing, upgrading, configuring, and removing computer programs for a computer's operating system in a consistent manner
  - A package management system
- Linux
  - APT (Advanced Package Tool) is a software user interface for installing and removing software on Debian, ubuntu, and other Linux distributions
    - Part of the Debian Package Management System
  - yum is the primary tool for getting, installing, deleting, querying, and managing Red Hat Enterprise Linux RPM software packages from official Red Hat software repositories
  - https://www.digitalocean.com/community/tutorials/package-management-basics-apt-yum-dnf-pkg

Homebrew 

- The “missing” Package Manager for Mac OS
  - It’s really just Ruby and git
  - https://brew.sh
- Fun fact: Max Howell, the creator of Homebrew was rejected for a position at google.
  - @mxcl: “Google: 90% of our engineers use the software you wrote (Homebrew), but you can’t invert a binary tree on a whiteboard so f*** off.”

pip and conda

- **Conda vs. pip vs. virtualenv commands**
- If you have used pip and virtualenv in the past, you can use conda to perform all of the same operations.
- Pip is a package manager, and virtualenv is an environment manager. Conda is both.

- **pip** (Python Installs Packages) is a package management system used to install and manage software packages written in Python
  - User guide: https://pip.pypa.io/en/stable/user_guide/
- **Conda** is an open source package management system and environment management system designed by Anaconda, Inc. and included in all Anaconda software distributions
  - User guide: https://conda.io/docs/user-guide/index.html

Pip install

Source Distributions vs Wheels

- pip can install from either Source Distributions (sdist) or Wheels, but if both are present on PyPI, pip will prefer a compatible wheel.
- Wheels are a pre-built distribution format that provides faster installation compared to Source Distributions (sdist), especially when a project contains compiled extensions.
- If pip does not find a wheel to install, it will locally build a wheel and cache it for future installs, instead of rebuilding the source distribution in the future.

DataCamp Anaconda/Conda Courses

Conda Essentials

- How use the Conda package manager to create and share reproducible environments for data science development.

Conda for Building and Distributing Packages

- How to create a "Conda Project" - a data science asset that specifies package installs, file downloads, and executable commands. Anaconda projects can be used to run Jupyter notebooks, Bokeh server apps, REST APIs, and command line tools on Windows, Mac OSX, and Linux platforms making deployment easy.

Checking your version of conda

- You can check your version of conda by running:
  - `$ conda --version`
  - `$ conda –V`

- To see all available options (flags)
  - `$ conda --help`

Conda/Pip/Virtualenv equivalents

Task	Conda package and environment manager command	Pip package manager command	Virtualenv environment manager command
Install a package	`conda install $PACKAGE_NAME`	`pip install $PACKAGE_NAME`	X
Update a package	`conda update --name $ENVIRONMENT_NAME $PACKAGE_NAME`	`pip install --upgrade $PACKAGE_NAME`	X
Update package manager	`conda update conda`	`Linux/macOS: pip install -U pip Win: python -m pip install -U pip`	X
Uninstall a package	`conda remove --name $ENVIRONMENT_NAME $PACKAGE_NAME`	`pip uninstall $PACKAGE_NAME`	X
Create an environment	`conda create --name $ENVIRONMENT_NAME python`	X	`cd $ENV_BASE_DIR; virtualenv $ENVIRONMENT_NAME`
Activate an environment	`source activate $ENVIRONMENT_NAME`	X	`source $ENV_BASE_DIR/$ENVIRONMENT_NAME/bin/activate`
Deactivate an environment	`source deactivate`	X	`deactivate`
Search available packages	`conda search $SEARCH_TERM`	`pip search $SEARCH_TERM`	X
Install package from specific source	`conda install --channel $URL $PACKAGE_NAME`	`pip install --index-url $URL $PACKAGE_NAME`	X
List installed packages	`conda list --name $ENVIRONMENT_NAME`	`pip list`	X
Create requirements file	`conda list --export`	`pip freeze`	X
List all environments	`conda info --envs`	X	`Install virtualenv wrapper, then lsvirtualenv`
Install other package manager	`conda install pip`	`pip install conda`	X
Install Python	`conda install python=x.x`	X	X
Update Python	`conda update python*`	X	X

Check to see what is available through conda

To see all available conda commands

- `$ conda –help`

```
DESCRIPTION
       usage: conda help [-h] [COMMAND]

       Displays a list of available conda commands and their help strings.

OPTIONS
   positional arguments:
       COMMAND
              Print help information for COMMAND (same as: conda COMMAND
              --help).

   optional arguments:
       -h, --help
              Show this help message and exit.

EXAMPLES
              conda help install
```

```
% conda ––help
usage: conda [-h] [-V] command ...

conda is a tool for managing and deploying applications, environments and packages.

Options:

positional arguments:
  command
    clean        Remove unused packages and caches.
    config       Modify configuration values in .condarc. This is modeled
                 after the git config command. Writes to the user .condarc
                 file (/Users/brentskoumal/.condarc) by default.
    create       Create a new conda environment from a list of specified
                 packages.
    help         Displays a list of available conda commands and their help
                 strings.
    info         Display information about current conda install.
    install      Installs a list of packages into a specified conda
                 environment.
    list         List linked packages in a conda environment.
    package      Low-level conda package utility. (EXPERIMENTAL)
    remove       Remove a list of packages from a specified conda environment.
    uninstall    Alias for conda remove. See conda remove ––help.
    search       Search for packages and display associated information. The
                 input is a MatchSpec, a query language for conda packages.
                 See examples below.
    update       Updates conda packages to the latest compatible version. This
                 command accepts a list of package names and updates them to
                 the latest versions that are compatible with all other
                 packages in the environment. Conda attempts to install the
                 newest versions of the requested packages. To accomplish
                 this, it may update some packages that are already installed,
                 or install additional packages. To prevent existing packages
                 from updating, use the ––no-update-deps option. This may
                 force conda to install older versions of the requested
                 packages, and it does not prevent additional dependency
                 packages from being installed. If you wish to skip dependency
                 checking altogether, use the '––force' option. This may
                 result in an environment with incompatible packages, so this
                 option must be used with great caution.
    upgrade      Alias for conda update. See conda update ––help.

optional arguments:
  -h, ––help     Show this help message and exit.
  -V, ––version  Show the conda version number and exit.

conda commands available from other packages:
  build
  convert
  develop
  env
  index
  inspect
  metapackage
  render
  server
  skeleton
  verify
```

Display information about current conda install

- `$ conda info`

```
DESCRIPTION
       usage:  conda info [-h] [--json] [--debug] [--verbose] [--offline] [-a]
       [-e]

              [-l] [-s] [--base] [--unsafe-channels] [packages [packages ...]]

       Display information about current conda install.

OPTIONS
   positional arguments:
       packages
              Display information about packages.

   optional arguments:
       -h, --help
              Show this help message and exit.

       --json Report all output as json. Suitable for using conda programmati-
              cally.

       --debug
              Show debug output.

       --verbose, -v
              Use once for info, twice for debug, three times for trace.

       --offline
              Offline mode, don't connect to the Internet.

       -a, --all
              Show  all information, (environments, license, and system infor-
              mation.

       -e, --envs
              List all known conda environments.

       -l, --license
              Display information about the local conda licenses list.

       -s, --system
              List environment variables.

       --base Display base environment path.

       --unsafe-channels
              Display list of channels with tokens exposed.

EXAMPLES
              conda info -a
```

```
% conda info

     active environment : None
       user config file : /Users/brentskoumal/.condarc
 populated config files : /Users/brentskoumal/.condarc
          conda version : 4.5.11
    conda-build version : 3.11.0
         python version : 3.6.6.final.0
       base environment : /anaconda3  (writable)
           channel URLs : https://conda.anaconda.org/conda-forge/osx-64
                          https://conda.anaconda.org/conda-forge/noarch
                          https://repo.anaconda.com/pkgs/main/osx-64
                          https://repo.anaconda.com/pkgs/main/noarch
                          https://repo.anaconda.com/pkgs/free/osx-64
                          https://repo.anaconda.com/pkgs/free/noarch
                          https://repo.anaconda.com/pkgs/r/osx-64
                          https://repo.anaconda.com/pkgs/r/noarch
                          https://repo.anaconda.com/pkgs/pro/osx-64
                          https://repo.anaconda.com/pkgs/pro/noarch
          package cache : /anaconda3/pkgs
                          /Users/brentskoumal/.conda/pkgs
       envs directories : /anaconda3/envs
                          /Users/brentskoumal/.conda/envs
               platform : osx-64
             user-agent : conda/4.5.11 requests/2.19.1 CPython/3.6.6 Darwin/18.0.0 OSX/10.14
                UID:GID : 501:20
             netrc file : None
           offline mode : False
```

List all packages in a conda environment

- `$ conda list`

- Because conda installs packages automatically, it's hard to know which package versions are actually on your system. b/c packages you didn't install explicitly get installed for you to resolve another package's dependencies.

```
DESCRIPTION
       usage: conda list [-h] [-n ENVIRONMENT | -p PATH] [--json] [--debug]

              [--verbose]  [--show-channel-urls] [--no-show-channel-urls] [-c]
              [-f] [--explicit] [--md5] [-e] [-r] [--no-pip] [regex]

       List linked packages in a conda environment.

OPTIONS
   positional arguments:
       regex  List only packages matching this regular expression.

   optional arguments:
       -h, --help
              Show this help message and exit.

       -n ENVIRONMENT, --name ENVIRONMENT
              Name of environment.

       -p PATH, --prefix PATH
              Full path to environment prefix.

       --json Report all output as json. Suitable for using conda programmati-
              cally.

       --debug
              Show debug output.

       --verbose, -v
              Use once for info, twice for debug, three times for trace.

       --show-channel-urls
              Show  channel  urls.  Overrides the value given by `conda config
              --show show_channel_urls`.

       --no-show-channel-urls
              Don't show channel urls. Overrides the  value  given  by  `conda
              config --show show_channel_urls`.

       -c, --canonical
              Output canonical names of packages only. Implies --nopip.

       -f, --full-name
              Only search for full names, i.e., ^<regex>$.

       --explicit
              List  explicitly  all  installed conda packaged with URL (output
              may be used by conda create --file).

       --md5  Add MD5 hashsum when using --explicit

       -e, --export
              Output requirement string only (output may be used by conda cre-
              ate --file).

       -r, --revisions
              List the revision history and exit.

       --no-pip
              Do not include pip-only installed packages.

EXAMPLES
       List all packages in the current environment:

              conda list

       List all packages installed into the environment 'myenv':

              conda list -n myenv

       Save packages for future use:

              conda list --export > package-list.txt
```

```
% conda list
# packages in environment at /anaconda3:
#
# Name                    Version                   Build  Channel
_ipyw_jlab_nb_ext_conf    0.1.0                     py36_0
absl-py                   0.4.1                      py_0    conda-forge
alabaster                 0.7.11                     py_3    conda-forge
anaconda                  custom            py36ha4fed55_0
anaconda-client           1.7.1                      py_0    conda-forge
anaconda-navigator        1.8.7                     py36_0
anaconda-project          0.8.2                      py_1    conda-forge
appnope                   0.1.0                     py36_0    conda-forge
appscript                 1.0.1             py36h470a237_0    conda-forge
asn1crypto                0.24.0                    py36_3    conda-forge
astor                     0.7.1                      py_0    conda-forge
astroid                   2.0.2                     py36_0    conda-forge
astropy                   3.0.4                     py36_0    conda-forge
atomicwrites              1.2.1                     py36_0    conda-forge
attrs                     18.2.0                     py_0    conda-forge
babel                     2.6.0                      py_1    conda-forge
backcall                  0.1.0                      py_0    conda-forge
backports                 1.0                        py_2    conda-forge
backports.shutil_get_terminal_size 1.0.0                       py_3    conda-forge
backports.weakref         1.0.post1                 py36_0    conda-forge
beautifulsoup4            4.6.3                     py36_0    conda-forge
bitarray                  0.8.3             py36h470a237_0    conda-forge
bkcharts                  0.2                       py36_0    conda-forge
blas                      1.0                          mkl
blaze                     0.11.3                    py36_0    conda-forge
bleach                    2.1.4                      py_1    conda-forge
blinker                   1.4                        py_1    conda-forge
blosc                     1.14.4               hfc679d8_0    conda-forge
bokeh                     0.13.0                    py36_0    conda-forge
boto                      2.49.0                    py36_0
boto3                     1.9.4                      py_0    conda-forge
botocore                  1.12.4                     py_0    conda-forge
bottleneck                1.2.1             py36h7eb728f_1    conda-forge
bz2file                   0.98                      py36_0
bzip2                     1.0.6                          1    conda-forge
c-ares                    1.14.0               h470a237_0    conda-forge
ca-certificates           2018.8.24            ha4d7672_0    conda-forge
certifi                   2018.8.24                 py36_1    conda-forge
cffi                      1.11.5            py36h5e8e0c9_1    conda-forge
chardet                   3.0.4                     py36_3    conda-forge
clangdev                  6.0.1                default_1    conda-forge
click                     6.7                        py_1    conda-forge
cloudpickle               0.5.6                      py_0    conda-forge
clyent                    1.2.2                      py_1    conda-forge
colorama                  0.3.9                      py_1    conda-forge
conda                     4.5.11                    py36_0    conda-forge
```

Semantic Versioning

- Under **semantic versioning**, software is labeled with a three-part version identifier of the form MAJOR.MINOR.PATCH
  - the label components are non-negative integers separated by periods. Assuming all software starts at version 0.0.0
  - The MAJOR version number is increased when significant new functionality is introduced (often with corresponding API changes)
  - The MINOR version number is increased when improvements (e.g., new features) are made that avoid backward-incompatible API changes.
  - The PATCH version number is increased when bug fixes are made that preserve the same MAJOR and MINOR revision numbers.

Installing packages

- Installing packages is easy:
- `$ conda install <pkg>`
- What is happening behind the scenes though?
  - The versions of packages to install (along with all their dependencies) must be compatible with all versions of other software currently installed.
  - Conda is special among package managers in that it always guarantees this consistency
    - you will see the phrase "Solving environment: " during installation which indicates this process

Installing packages

- What is happening behind the scenes though?
  - The versions of packages to install (along with all their dependencies) must be compatible with all versions of other software currently installed.
  - Conda is special among package managers in that it always guarantees this consistency
    - you will see the phrase "Solving environment: " during installation which indicates this process

```
% conda install —help                                                    ~
usage: conda install [-h] [—revision REVISION] [-y] [—dry-run] [-f]
                     [—file FILE] [—no-deps] [—only-deps] [-m] [-C]
                     [—use-local] [—offline] [—no-pin] [-c CHANNEL]
                     [—override-channels] [-n ENVIRONMENT | -p PATH] [-q]
                     [—copy] [-k] [—update-dependencies]
                     [—no-update-dependencies] [—channel-priority]
                     [—no-channel-priority] [—clobber] [—show-channel-urls]
                     [—no-show-channel-urls] [—download-only] [—json]
                     [—debug] [—verbose]
                     [package_spec [package_spec ...]]

Installs a list of packages into a specified conda environment.

    This command accepts a list of package specifications (e.g, bitarray=0.8)
    and installs a set of packages consistent with those specifications and
    compatible with the underlying environment. If full compatibility cannot
    be assured, an error is reported and the environment is not changed.

    Conda attempts to install the newest versions of the requested packages. To
    accomplish this, it may update some packages that are already installed, or
    install additional packages. To prevent existing packages from updating,
    use the —no-update-deps option. This may force conda to install older
    versions of the requested packages, and it does not prevent additional
    dependency packages from being installed.

    If you wish to skip dependency checking altogether, use the '—force'
    option. This may result in an environment with incompatible packages, so
    this option must be used with great caution.

    conda can also be called with a list of explicit conda package filenames
    (e.g. ./lxml-3.2.0-py27_0.tar.bz2). Using conda in this mode implies the
    —force option, and should likewise be used with great caution. Explicit
    filenames and package specifications cannot be mixed in a single command.
```

Updating packages

- `$ conda update <pkg>`
- Closely related to installing a particular version of a conda package is updating the installed version to the latest version possible that remains compatible with other installed software. conda will determine if it is possible to update dependencies of the package(s) you are directly updating, and do so if resolvable.

```
% conda update numpy
Solving environment: done

## Package Plan ##

  environment location: /anaconda3

  added / updated specs:
    - numpy


The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    scikit-learn-0.20.0        |py36_blas_openblash00c3548_201        5.5 MB  conda-forge
    numpy-1.15.4               |py36_blas_openblashb06ca3d_0          4.1 MB  conda-forge
    openblas-0.3.3             |       ha44fe06_1        16.7 MB  conda-forge
    numpy-base-1.15.4          |    py36ha711998_0         4.0 MB
    scipy-1.1.0                |py36_blas_openblashb06ca3d_202       15.5 MB  conda-forge
    ------------------------------------------------------------
                                           Total:        45.9 MB

The following packages will be UPDATED:

    blas:         1.0-mkl                        --> 1.1-openblas                          conda-forge
    numpy:        1.15.1-py36h6a91979_0          --> 1.15.4-py36_blas_openblashb06ca3d_0   conda-forge [blas_openblas]
    numpy-base:   1.15.1-py36h8a80b8c_0          --> 1.15.4-py36ha711998_0
    openblas:     0.2.20-8             conda-forge --> 0.3.3-ha44fe06_1                      conda-forge
    scikit-learn: 0.19.1-py36hffbff8c_0          --> 0.20.0-py36_blas_openblash00c3548_201 conda-forge [blas_openblas]
    scipy:        1.1.0-py36h28f7352_1           --> 1.1.0-py36_blas_openblashb06ca3d_202  conda-forge [blas_openblas]

Proceed ([y]/n)?
```

Removing packages

- `$ conda remove <pkg>`
- As with other commands, you may also optionally specify multiple packages separated by spaces.
- Note that conda always tries to use the most recent versions of installed software that are compatible. Therefore, sometimes removing one package allows another package to be upgraded implicitly because only the removed package was requiring the older version of the dependency.

```
% conda remove six
Solving environment: done

## Package Plan ##

  environment location: /anaconda3

  removed specs:
    - six


The following packages will be REMOVED:

    _ipyw_jlab_nb_ext_conf: 0.1.0-py36_0
    absl-py:                0.4.1-py_0           conda-forge
    anaconda-client:        1.7.1-py_0           conda-forge
    anaconda-navigator:     1.8.7-py36_0
    anaconda-project:       0.8.2-py_1           conda-forge
    astroid:                2.0.2-py36_0         conda-forge
    astropy:                3.0.4-py36_0         conda-forge
    bkcharts:               0.2-py36_0           conda-forge
    blaze:                  0.11.3-py36_0        conda-forge
    bleach:                 2.1.4-py_1           conda-forge
    bokeh:                  0.13.0-py36_0        conda-forge
    boto3:                  1.9.4-py_0           conda-forge
    botocore:               1.12.4-py_0          conda-forge
    conda:                  4.5.11-py36_1000     conda-forge
    conda-build:            3.11.0-py36_0        conda-forge
    cryptography:           2.3.1-py36hdffb7b8_0 conda-forge
    cycler:                 0.10.0-py_1          conda-forge
    dask:                   0.19.1-py_0          conda-forge
    datashape:              0.5.4-py36_0         conda-forge
    distributed:            1.23.1-py36_0        conda-forge
    flask-cors:             3.0.6-py_0           conda-forge
    gensim:                 3.4.0-py36h1de35cc_0
    grpcio:                 1.14.1-py36hd60e7a3_0 conda-forge
    h5py:                   2.8.0-py36hb794570_1 conda-forge
    html5lib:               1.0.1-py_0           conda-forge
    ipykernel:              4.9.0-py36_0         conda-forge
    ipython:                6.5.0-py36_0         conda-forge
    ipywidgets:             7.4.2-py_0           conda-forge
    jupyter:                1.0.0-py_1           conda-forge
    jupyter_client:         5.2.3-py_1           conda-forge
```

Conda channels

- All Conda packages we've seen so far were published on the main or default channel of Anaconda Cloud.
- A *Conda channel* is an identifier of a path (e.g., as in a web address) from which Conda packages can be obtained. Using the public cloud, installing without specifying a channel points to the main channel at **https://repo.continuum.io/pkgs/main**; where hundreds of packages are available.
- Anyone may register for an account with Anaconda Cloud, thereby creating their own personal Conda channel.
- This is covered in the datacamp course Conda for Building and Distributing Packages (along with creating and uploading your own packages).
- For this course, just understand that many users have accounts and corresponding channels.

Special Conda channels

```
$ conda config --add channels conda-forge
```

- **Default, non-default, and special channels**
- The default channel on Anaconda Cloud is curated by Anaconda Inc., but another channel called conda-forge also has a special status. This channel does not operate any differently than other channels, whether those others are associated with an individual or organization, but it acts as a kind of "community curation" of relatively well-vetted packages.
- The GitHub page for the conda-forge project at **https://github.com/conda-forge** describes it as: "A community led collection of recipes, build infrastructure and distributions for the conda package manager."
- Apart from the somewhat more organized conda-forge channel/project, Anaconda Cloud channels are relatively anarchic. Much like GitHub repos or packages on the Python Package Index (PyPI), anyone is free to upload whatever projects they like to conda-forge (as long as they are assembled as Conda packages, that is, but this is a minor restriction).

Searching for packages across channels

- **Searching across channels**
- Although the conda command and its subcommands are used for nearly everything in this course, the package anaconda-client provides the command anaconda that searches in a different manner that is often more useful.
- For instance, you may know the name of the textadapter package, but you may not know in which channel (or channels) it may be published (or by which users). You can search across *all channels* and *all platforms* using:
  - `$ anaconda search textadapter`

Why use a virtual/conda environment?

- Conda *environments* allow multiple incompatible versions of the same (software) package to coexist on your system.
  - An *environment* is simply a filepath containing a collection of mutually compatible packages. By isolating distinct versions of a given package (and their dependencies) in distinct environments, those versions are all available to work on particular projects or tasks.
- There are a large number of reasons why it is best practice to use environments, whether as a data scientist, software developer, or domain specialist.
  - Without the concept of environments, users essentially rely on and are restricted to whichever particular package versions are installed globally (or in their own user accounts) on a particular machine. Even when one user moves scripts between machines (or shares them with a colleague), the configuration is often inconsistent in ways that interfere with seamless functionality. Conda environments solve both these problems. You can easily maintain and switch between as many environments as you like, and each one has exactly the collection of packages that *you* want.
- For example, you may develop a project comprising scripts, notebooks, libraries, or other resources that depend on a particular collection of package versions. You later want to be able to switch flexibly to newer versions of those packages and to ensure the project continues to function properly before switching wholly. Or likewise, you may want to share code with colleagues who are required to use certain package versions. In this context, an environment is a way of documenting a known set of packages that correctly support your project.

Which environment am I using?

- When using conda, you are always in some environment, but it may be the default (called the *base* or *root* environment).
- Your current environment has a name and contains a collection of packages currently associated with that environment. There are a few ways to determine the current environment.
  - 1) The name of the current environment is usually prepended to the rest of your prompt in parentheses.
  - 2) The command $ **conda env list** displays a list of all environments on your current system
    - the currently activated one is marked with an asterisk in the middle column.

Which environment am I using?

- The output of conda env list shows that each environment is associated with a *particular directory*.
  - This is **not** the same as your *current working directory* for a given project
  - being "in" an environment is completely independent of the directory you are working in.
  - The environment directory displayed by **conda env list** is simply the top-level filepath in which all resources associated with that environment are stored

What packages are installed in my environment?

- It's very simple to see the packages (and associated versions and builds) installed in each of your conda environments:
  - $ conda list
- We don't even have to activate the conda environment to list the installed packages:
  - $ conda list --name env_name <pkg_name> …

How do I actually use these environments

- You can *activate* an environment with the following command
  - $ source activate env_name (Linux, OSX)
  - $ activate env_name (Windows)
  - $ conda activate (conda 4.4.0)
- You can *deactivate* an environment with the following command
  - $ source deactivate env_name (Linux, OSX)
  - $ deactivate env_name (Windows)
  - $ conda deactivate (conda 4.4.0)

Creating/Removing environments

- You can create environments a few different ways:
  - Just name & default python: $ conda create --name env_name
  - Name & different python version: $ conda create --name env_name python=2.7
  - Name, version, packages: $ conda create --name env_name python=2.7 pandas
- You can also get rid of unwanted environments:
  - $ conda env remove –n env_name

Saving your environment specs

- There are a few ways to save the specs of your environment:
- 1) List out the package name, version, and build and create a requirements.tx file:
  - $ conda list –e > requirements.txt
- 2) Export your environment to a YAML file with conda:
  - $ conda env export --name env_name --file file_name

Creating an environment from a spec file

- If you use a requirements.txt file:
  - $ conda create --name env_name --file requirements.txt
- If you are using an environment.yml file:
  - $ conda env create --name env_name --file env_file.yml

Example – Packaging your code