# What is an Anaconda Project?

In the Conda Essentials course you learned how use the Conda package manager to create and share reproducible environments for data science development.

In this chapter you'll create an [Anaconda Project](https://anaconda-project.readthedocs.io/en/latest), which is 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.

Anaconda projects are shared amongst data scientists as compressed directories containing the Conda environment specification, URLs for downloadable files, and source code for commands. Collaboration can be achieved through text file revision control tools, like git.

In the following exercises you'll learn how to create, run, and share Anaconda Projects.

Which of the following CANNOT be achieved with an Anaconda Project?

##### Answer the question

##### Possible Answers

* Develop a project on your workstation and deploy it to a production web server.

press

* Automatically install the correct version of each required package before running a command.

press

* Provide an on-line editing environment for multiple users to develop code.

Press **(A)**

* Define datasets to be downloaded before running commands.

press

# **Install Anaconda-Project**

The Anaconda Project executable you'll need to create, run, and share projects is provided as a conda package.

First we need to make sure that it has been installed.

##### Instructions

100 XP

Install the anaconda-project package.

**ANS :** conda install anaconda-project

# Prepare and run a project command

An Anaconda Project is provided for you at the directory /home/repl/babynames. There you will see the anaconda-project.yml file, which is the project specification file, and a Python script file main.py. When collaborating with other data scientists you may receive Anaconda Projects as compressed directories containing an anaconda-project.yml file along with source files.

In these exercises you'll explore this project and run the command-line-interface (CLI) defined in main.py.

The dataset used here is the number of male and female babes born by name in the US from 2000 to 2016 provided by [The Social Security Administration](https://catalog.data.gov/dataset/baby-names-from-social-security-card-applications-national-level-data).

##### Instructions 1/4

25 XP

* [1](javascript:void(0))

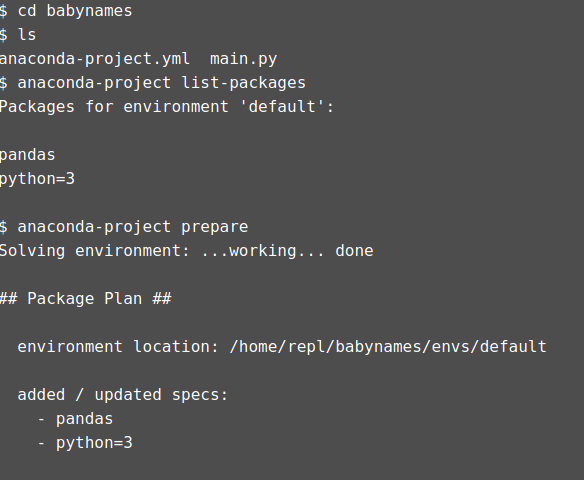
The first step is to determine what packages are required for this project. For this you'll inspect the output of anaconda-project list-packages. This will show you what Conda packages are required to run this project.

**ANS : anaconda-project list-packages**

Now that you know that this project requires Python version 3 and the latest version of Pandas, you can prepare a new Conda environment within the project.

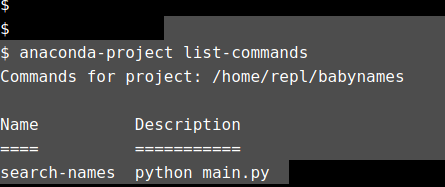
Run anaconda-project prepare to create the environment. It will create an envs/default directory in your project.

**ANS : anaconda-project prepare**

****

This project contains a main.py Python script file. To determine if the project itself has a defined command we'll look at the output of anaconda-project list-commands.

ANS :



Running a command with Anaconda Project will automatically activate the conda environment that was created in envs/default.

The command search-names, defined in main.py, has the following help output.

usage: search-names [-h] name

Print occurrences of a name yearly since 2000.

positional arguments:

name Name to be searched for

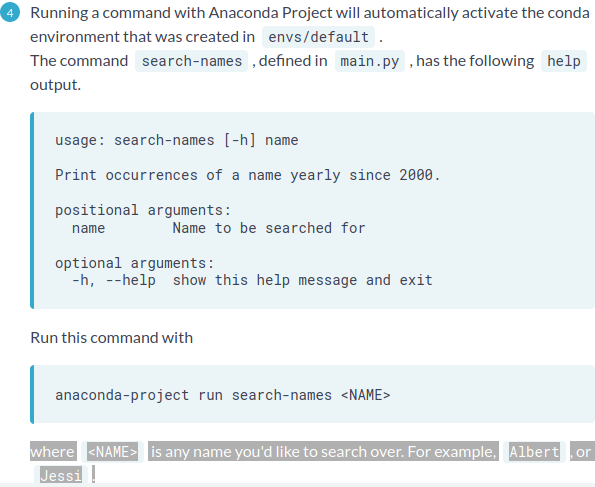
optional arguments:

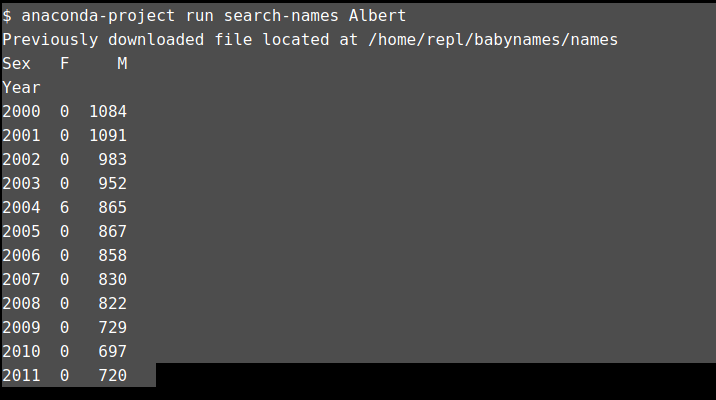
-h, --help show this help message and exit

Run this command with

anaconda-project run search-names <NAME>

where <NAME> is any name you'd like to search over. For example, Albert, or Jessi.





# Anaconda Project specification file

The core of an Anaconda Project is a YAML file containing a specification of the conda packages, commands, and downloads that make up the Project.

The YAML file is called anaconda-project.yml and each separate project you create will be in it's own subdirectory containing a distinct anaconda-project.yml file for that project.

In the terminal you will see that we have navigated to the babynames directory. Use ls to inspect the contents of this directory. Further, inspect the anaconda-project.yml file. You should see YAML tags for packages, commands, and downloads.

Choose the correct command below that was used when you executed anaconda-project run search-names <NAME> in the previous exercise. You can use tools like nano, vim, emacs, cat, less, or more to read the file.

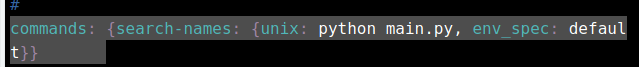
Instructions

50 XP

##### Possible Answers

* search-names.exe
* python main.py **(A)**
* python search-names
* main.py





# Initialize a new project

Now it's your turn to create an Anaconda Project. In the following exercises you'll create a new directory to store your project, initialize the anaconda-project.yml specification file, add packages, and prepare the environment.

This project will download [historic 30-year mortgage rates from the St. Louis Federal Reserve](https://fred.stlouisfed.org/series/MORTGAGE30US), fit an autoregressive model using data form January 2000 to January 2018, and forecast the mortgage rates monthly through December 2018.

##### Instructions 1/4

25 XP

* [1](javascript:void(0))

Create a new directory called mortgage\_rates.

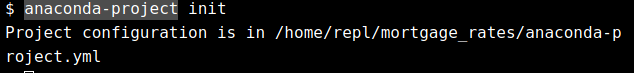


Navigate to the mortgage\_rates directory.



Initialize the Anaconda Project specification using the following command:

anaconda-project init



Many of the actions you'll perform on your Anaconda Projects do not require manual editing of the anaconda-project.yml, but doing so can be very useful.

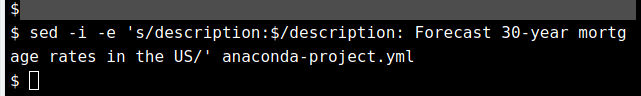
One action that does not have a CLI tool is adding a description of your project.

Edit the anaconda-project.yml file to add a description of the project. The description: tag is on line 19 of the file.

Edit the file so that line 19 looks like this

description: Forecast 30-year mortgage rates in the US

Available editors are nano, vi, vim, and emacs.



# Anaconda Project commands

As you have seen Anaconda Projects provide reproducible execution of data science assets. The babynames project defined a command-line-interface (CLI) command to analyze yearly trends.

When defining a command line tool in Anaconda Project the type unix or windows should be used. Typically both are defined, where unix is the full command as run in Bash and windows is the full command as run in the Windows Shell.

Anaconda Projects can support four types of commands:

* Unix commands: shell-based commands that run on Mac or Unix systems
* Windows commands: Windows shell commands
* Bokeh App: Run bokeh-server with a given Python script
* Jupyter Notebook: Launch Jupyter Notebook with the specified notebook file

For both Unix and Windows commands any arbitrary command can be run. These could be OS-specific tools or Python scripts provided with the Project. The Conda environment defined in anaconda-project.yml is created and activated automatically when running a command.

Commands are added to projects using the anaconda-project add-command command. Projects can have any number of commands defined.

Which of the following tasks are not supported by Anaconda Project commands?

##### Answer the question

50 XP

##### Possible Answers

* Launch a rich web dashboard built with Bokeh

press

* Start a light-weight REST API built with Python

press

* Launch a graphical user interface (GUI) tool

press

* None of the above **(A)**

# **Add packages and commands**

Now the you have initialized the mortgage\_rates project, it's time to add packages, downloads, and commands.

##### Instructions 1/5

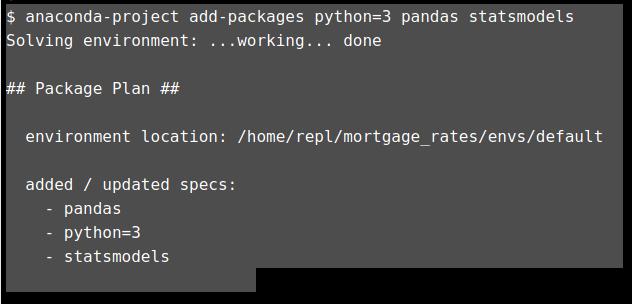
20 XP

* [1](javascript:void(0))

Now it's time to add the required packages to the project and prepare the environment. This Python 3 based project will need to install Pandas for data processing and statsmodels to fit the autoregressive model and make predictions.

All other dependencies of these packages, like NumPy and Scipy will be installed automatically. Run anaconda-project add-packages <package-names> with the following

* + python=3
  + pandas
  + statsmodels

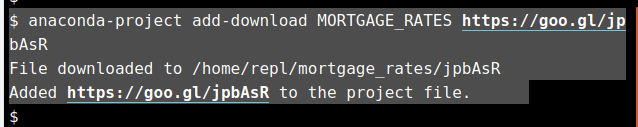


Now it's time to add the URL to download the 30-year mortgage rates. In a project, files are downloaded automatically when a command is run. A download is defined through the add-download command which takes two arguments.

anaconda-project add-download <ENV\_VARIABLE> <DOWNLOAD\_URL>

The first argument <ENV\_VARIABLE> is the shell variable name for the downloaded file and the second argument <DOWNLOAD\_URL> is the full URL to the file.

Your job is to add a download for <https://goo.gl/jpbAsR> with the variable name MORTGAGE\_RATES. The link has been shortened for convenience.



In order to use the mortgage rates CSV file that will be automatically downloaded by Anaconda Project we have to access the MORTGAGE\_RATES environment variable from our Python code. The file forecast.py is now in your mortgage\_rates project directory. Edit the file so that line 9 reads

MORTGAGE\_RATES = os.environ["MORTGAGE\_RATES"]

This line uses the os Python module to access environment variables from the system shell.

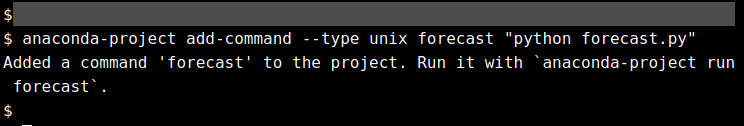
You can use nano, vi, vim, or emacs to the edit the file.

**ANS : sed -i -e 's/MORTGAGE\_RATES = \.\.\./MORTGAGE\_RATES = os.environ["MORTGAGE\_RATES"]/' forecast.py**

The script file is now ready and the next step is to register it as a command in your project. The script file is intended to be run as a command line tool and must be declared a Unix command with --type unix. Here's the anaconda-project command.

anaconda-project add-command [flags] <name> <command-to-execute>

The <name> is forecast and the <command-to-execute> is "python forecast.py".



The project has now been completely specified and you can run the forecast command using anaconda-project run as you did before.



# Locking package versions

So far you've noticed that the mortgage\_rates project depends on the Python, Pandas, and Statsmodels packages, but we did not specify package versions other than that this project is written for Python version 3.x. This means that running anaconda-project prepare or anaconda-project run the first time will download and install the latest version of each package.

At any time you can request to update all packages using

anaconda-project update

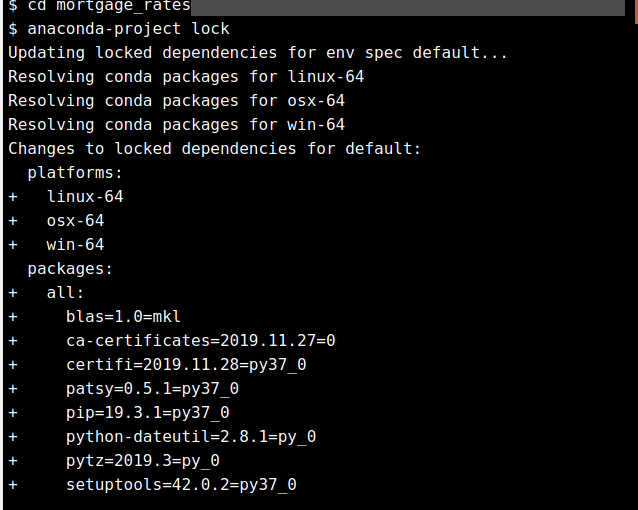
If you were to share this project with another data scientist they will quite likely get different versions of every package when they use it. Especially, if it has been several months since you sent them the package.

Use anaconda-project lock to write the current versions of every package, including low-level dependencies, for the Mac, Linux, and Windows platforms to the anaconda-project-lock.yml file so that other users can exactly recreate the environment as you were using today.

##### Instructions

50 XP

In the terminal run anaconda-project with the lock argument. You'll notice that exact package versions are printed for Mac, Linux, and Windows.



# Sharing your project

The core component of all Anaconda Projects is the anaconda-project.yml file. To share the project with other users you need to provide the anaconda-project.yml file and all other assets that cannot be automatically generated.

For the mortgage\_rates project you have been building, the three necessary files to recreate and run the project are anaconda-project.yml, anaconda-project-lock.yml, and forecast.py. The directory envs and the file jpbAsR should not be included in the archive.

In this exercise, you'll create a compressed project archive and share it on [Anaconda Cloud](https://anaconda.org/).

Note: You will need to [create a free account on Anaconda Cloud](_blank) to complete this exercise.

##### Instructions 1/3

30 XP

* [1](javascript:void(0))

Archive your mortgage\_rates project with the command

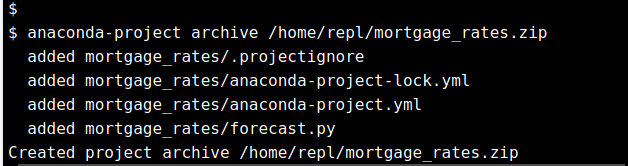
anaconda-project archive <zip-file>

where <zip-file> is the path to the zip file (in this case, /home/repl/mortgage\_rates.zip). Note that you can specify either the relative or absolute path.

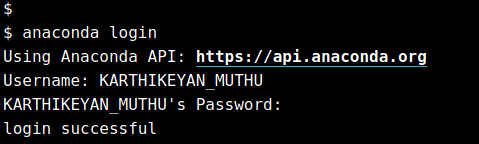
The zip file will only include anaconda-project.yml and forecast.py since those two files can be used to recreate the package and data downloads.

It is best practice not to create the zip archive within the project directory.

Notice that envs and jpbAsR are not included in the archive.



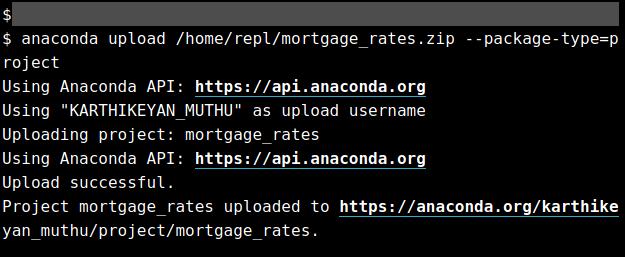
The next step is to login with your [Anaconda Cloud](https://anaconda.org/) account using anaconda login.



Now you're ready to upload the archive. Be certain to include --package-type= or -t to declare that this archive is an Anaconda Project.

anaconda upload <zip-file> --package-type=project

Where <zip-file> is the path to the archive you made in the last step.



# Python modules and packages

You are familiar with Python scripts that can be executed by running python <script-file>.py. Each of these files has one or more import statements to re-use Python source code written by other developers. These re-usable Python source files are referred to as modules. They are .py Python source files, like the scripts you have been writing, that are stored in a location where the import statement can find them.

There is no overtly recognizable difference between Python modules and scripts, except that developers of the former write them with the intention that their contents are imported, rather than executed by users.

In this chapter you'll learn how to turn your Python scripts into importable modules, collect those modules into packages and create Conda packages that can be easily installed by other users.

Which of the following is true about Python modules?

##### Answer the question

50 XP

##### Possible Answers

* Python modules must have the .pm file extension.

press

* Python modules must define code in abstract base classes.

press

* Python modules must be installed in order to use them. **(A)**

press

* Python modules cannot be executed using python <module-file>.

press

# Importing a module

You now know that any .py Python file can act as a module. A module is a Python file that can be imported into a Python terminal or script file. The first place Python will look for a module file is the current working directory. This means that any .py file you write can be used by other .py files in the same directory.

In your terminal there is a file called pi.py. Use tools like nano, vim, vi, less, more, or emacs to inspect the file. It defines a single function compute\_pi(). This function takes a single integer argument, n\_iters the number of iterations, larger values means you will get a more accurate estimate of *p**i*

but the computation will take longer.

Your job is to launch the Python interpreter by running python on the command line, import the pi module and call the compute\_pi() function. With input n\_iters = 1000 which value best matches what compute\_pi() returned?

##### Instructions

50 XP

##### Possible Answers

* 3
* 3.14159
* 3.14 **(A)**
* 2.9

# Modules and \_\_name\_\_

The single most important aspect of developing a Python module is to separate importable expressions from execution statements. When a user calls import on your module they generally want to use your functions rather than wait for lengthy computation or watch many rows of output be printed.

Your working directory now has a slightly different version of pi.py. Not only has the compute\_pi() function been defined but it also includes a new if statement:

if \_\_name\_\_ == '\_\_main\_\_':

pi = compute\_pi(int(8e6))

print('pi is approximately {:.7f}'.format(pi))

This statement allows the file to behave differently if it is imported or executed on the command line. When using import all rows above if \_\_name\_\_ == '\_\_main\_\_': are executed and imported; the print function will not be called.

Your job is to:

1. Execute pi.py on the command line
2. Start the Python interpreter and import pi

Did it take more time to import the file or execute the file?

##### Instructions

50 XP

##### Instructions

50 XP

##### Possible Answers

* The import statement was faster. **(A)**
* The execution was faster.
* The times were similar.

Correct!. Separating declarations from execution is the first step to make a module.

# Python package directory

So far you have seen that any .py Python source file can be used as a module to import code. A Python package is a collection of separate modules collected under a single name that share metadata, such as documentation, licensing, and version numbering.

Here's an example directory structure for my\_package. Note the repeated use of my\_package directories. The individual modules are stored in the inner my\_package directory.

my\_package/

LICENSE

README

setup.py

my\_package/

\_\_init\_\_.py

module1.py

module2.py

...

Once this package has been installed you can import code from the modules using the standard import idioms.

Over the next several exercises you're going to re-use much of the same code that was present in the mortgage\_rates Anaconda Project to prepare a Python package that can be installed and imported in new Python projects.

The setup.py script file coordinates installing the package into your Python distribution.

Why is it important to package re-usable Python code?

##### Answer the question

50 XP

##### Possible Answers

* To keep a consistent version history of changes.

press

* To make module installation and management easier for others.

press

* To clearly define importable API components.

press

* All of the above.  **(A)**

press

# Importing a package

A Python package can have any number of directories and module source files.

The directory mortgage\_forecasts has been prepared for you in your home directory. It contains a subdirectory of the same name and two Python source code files.

mortgage\_forecasts/

mortgage\_forecasts/

models.py

utils.py

models.py defines a new class to fit and predict 30-year mortgage rates in the US.

utils.py defines functions to read data and compute statistical quantities.

Your working directory has been set to /home/repl/mortgage\_forecasts. Without changing directories, which import statement will provide access to the MortgageRateModel class defined in models.py?

You can use the terminal to test import statements. A conda environment has been activated with the dependent packages.

##### Instructions

50 XP

##### Instructions

50 XP

##### Possible Answers

* import mortgage\_forecasts
* from mortgage\_forecasts.models import MortgageRateModel **(A)**
* from mortgage\_forecasts import MortgageRateModel
* import models.MortgageRateModel

# The \_\_init\_\_.py file

As you saw in the previous exercise you have to individually import models and utils to access functions and classes defined in each module. These modules do not get automatically imported when you run import mortgage\_forecasts, since mortgage\_forecasts is a directory.

To define what gets imported from a directory you need a file called \_\_init\_\_.py. This file is executed when import mortgage\_forecasts is run.

For the mortgage\_forecasts package we wish to provide a more convenient way to import MortgageRateModel from models.py and read\_data from utils.py. Our goal is to tell users of the package that they can access these tools as

from mortgage\_forecasts import read\_data, MortgageRateModel

The mortgage\_forecasts directory has been prepared for you.

##### Instructions 1/2

0 XP

* Create a new \_\_init\_\_.py file in /home/repl/mortgage\_forecasts/mortgage\_forecasts.

Add this docstring at the top of the file. This allows the user to run help(mortgage\_forcecasts) after importing the package.

'''Predictive modeling of 30-year mortgage rates in the US.'''

You can use nano, vi, vim, or emacs to edit the file.

**ANS :**

echo "'''Predictive modeling of 30-year mortgage rates in the US.'''" > /home/repl/mortgage\_forecasts/mortgage\_forecasts/\_\_init\_\_.py

Edit the \_\_init\_\_.py to add the following lines.

from .models import MortgageRateModel

from .utils import read\_data

Notice the use of . to mean the local directory of the \_\_init\_\_.py file. This helps avoid name collisions with other packages that have been installed.

**ANS :**

echo "'from .models import MortgageRateModel\nfrom .utils import read\_data" > /home/repl/mortgage\_forecasts/mortgage\_forecasts/\_\_init\_\_.py

# Create the installer script

At this point you have completed the source code development of the mortgage\_forecasts project. The next step is to write packaging scripts that will install the contents of the inner mortgage\_forecasts directory into your Python environment. The packaging scripts define metadata about your package like the author, the license, and what subdirectories contain the package code.

There are multiple methods to [package and deliver Python code](https://packaging.python.org/). In this course we'll cover a hybrid method that utilizes the [setuptools package](http://setuptools.readthedocs.io/en/latest/index.html) and [conda build recipes](https://conda.io/docs/user-guide/tasks/build-packages/index.html).

First, you're going to write a setup.py file. A template has been provided for you in the mortgage\_forecasts directory.

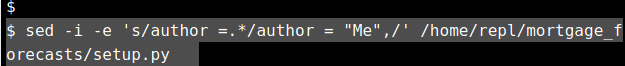
##### Instructions 1/2

50 XP

* [1](javascript:void(0))

Edit the setup.py file to add the following string arguments to the setup function call.

* + name: the package name mortgage\_forecasts
  + description: '30 year mortgage rate models'
  + author: your name



###### Solution

# Open any of the available editors to edit the file instead of using the following command

sed -i -e 's/name =.\*/name = "mortgage\_forecasts",/' /home/repl/mortgage\_forecasts/setup.py

sed -i -e 's/description =.\*/description = "30 year mortgage rate models",/' /home/repl/mortgage\_forecasts/setup.py

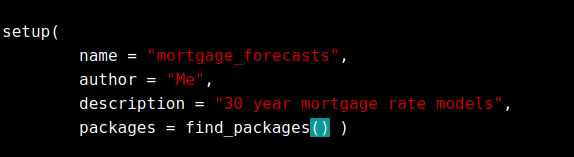
sed -i -e 's/author =.\*/author = "Me",/' /home/repl/mortgage\_forecasts/setup.py

**Q 2 :**

The final step is to specify what directories setuptools should use to find the package source files.

The find\_packages() function searches the directory where setup.py resides and returns a list of all subdirectories that have an \_\_init\_\_.py file.

Add packages = find\_packages() as an argument to setup.py.



# Licensing

Since our goal is to share our code with others we need to be aware of copyright laws and the legal rights we wish to retain about how that software can be used. Copyright protections are guaranteed to the person who owns the software. When someone else downloads and uses the program we built we would not want to transfer ownership to them, thereby forfeiting our rights.

Instead, we wish license usage of the program under certain restrictions. There are many [kinds of software licenses](https://en.wikipedia.org/wiki/Software_license) and choosing a license is beyond the scope of this course.

For the mortgage\_forecasts package we want to share our code with as few restrictions as necessary. The only important restriction is that we would like to be acknowledged as the original author. For this we'll choose the MIT license, which is widely used in open-source software packages.

The full text of the license has been placed in the mortgage\_forecasts/LICENSE file and it must remain in the package directory for the license to be valid and enforceable.

Further, license="MIT" has been added in setup.py.

which statement below is INCORRECT?

##### Instructions

50 XP

##### Instructions

50 XP

##### Possible Answers

* The MIT license does not allow commercial use. **(A)**
* The original developer retains ownership of the software.
* There is no fee paid to the developer to use this package.
* All copies of this package must retain the LICENCE file.

# Version number

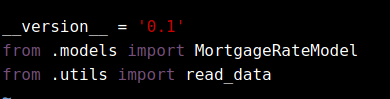
By convention Python version numbers are stored using the \_\_version\_\_ attribute of the package. While there are [several methods](https://packaging.python.org/guides/single-sourcing-package-version/) by which you can determine and write version numbers in your projects, in this exercise you're going to write it as a string in the \_\_init\_\_.py file.

The setup.py will read from \_\_init\_\_.py when the package is installed using the find\_version() function defined in version.py. This avoids having to define the version number in two places.

##### Instructions

100 XP

Edit the \_\_init\_\_.py file and assign \_\_version\_\_ to the string '0.1'.



# Install the package

Now you're ready to install the package for use. It's important to note that setup.py does not contain information about dependent packages, like pandas and statsmodels. While the setup.py file can define these packages we're going to use Conda Build in the next chapter to define and install dependent packages. For now in order to use the package we must have pandas, statsmodels, and scipy installed.

The forecast conda environment has been created and activated for you. In this exercise you'll install the mortgage\_forecasts package and verify that you can import and run it.

##### Instructions 1/3

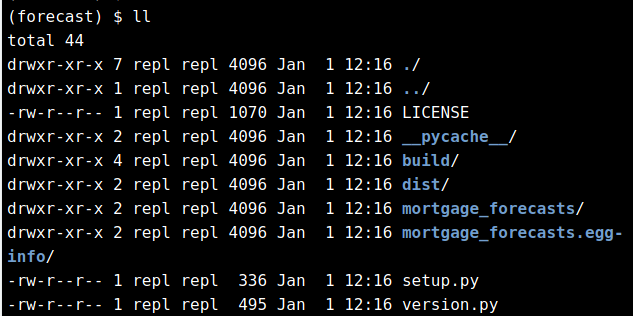
40 XP

* [1](javascript:void(0))

To install the package in your environment execute the setup.py file with install as the first argument.

Notice that the package is installed in the forecast Conda environment.

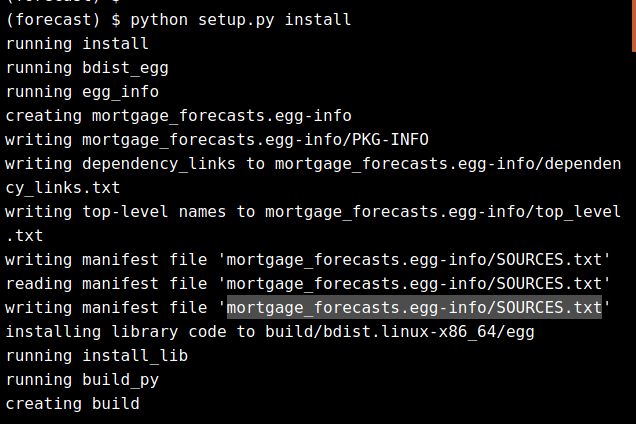
python setup.py install



In order to fit the MortgageRateModel autoregressive model you'll need to read mortgage rate data from the St. Louis Federal Reserve. The data has been downloaded for you.

Change directory to /home/repl/practice, where the data has been placed.

..



In order to fit the MortgageRateModel autoregressive model you'll need to read mortgage rate data from the St. Louis Federal Reserve. The data has been downloaded for you.

Change directory to /home/repl/practice, where the data has been placed.



In the Python shell import MortgageRateModel and read\_data from the mortgage\_forecasts package and call:

* read\_data() with the argument mortgage\_rates.csv -> df
* MortgageRateModel() with the argument df -> model
* model.forecast() with the argument 'January 2019'

You can continue to inspect the help() of the package, functions, and classes.

Exit the interpreter when you're done.

# Run python without any arguments to open the python shell instead of the following command

python -c "import mortgage\_forecasts"

# Conda Packages

In the last chapter you created a Python package and successfully installed it. However, you needed to have 1) downloaded the source code, and 2) created a Conda Environment with the dependent packages.

Further, when using setuptools to install packages there are no uninstall or update commands. That means you would have to manually remove the installed files if you want to install a newer version of the package. As you saw in the Conda Essentials course, Conda packages solve each of these issues, but you might use [pip and virtualenv](https://packaging.python.org/guides/installing-using-pip-and-virtualenv/) as well.

In this chapter you'll create a Conda Recipe for the mortgage\_forecasts package to define the dependent Conda packages. The Conda recipe is specified in a file called meta.yaml.

You'll then build the package archive and upload it to Anaconda Cloud. Further, Conda packages are not limited to Python packages. A package written in any programming language, or a collection of files, can become a Conda package.

Which statement below is INCORRECT?

##### Answer the question

50 XP

##### Possible Answers

* Having a setup.py file is enough to build a Conda package. **(A)**

press

* Conda packages can be built for JavaScript and R source packages.

press

* Dependent conda packages are specified in the meta.yaml file.

press

* python setup.py update works just like conda update <package>

press

Correct! You need the conda recipe meta.yaml file.

# Install Conda Build

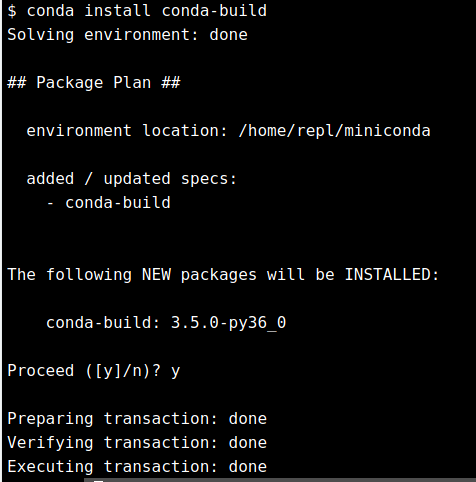
In order to build a Conda package you first need to install the conda-build package.

The conda build command will read the Conda recipe and construct a package file that can be uploaded to [Anaconda Cloud](https://anaconda.org/) and installed in a Conda environment.

##### Instructions

100 XP

Install conda-build into your current Conda environment.



# The Conda recipe meta.yaml

The mortgage\_forecasts package you wrote in the last chapter has been provided in your home directory along with a template meta.yaml Conda recipe. There are 5 sections of this file:

* package defines the package name and version
* source provides the relative path (or Github repository) to the package source files
* build defines the command to install the package
* requirements specify the conda packages required to build and run the package
* about provides other important metadata like the license and description

Inspect the meta.yaml file with nano, vim, emacs, cat, less or more.

You'll see at the top of the file {% set setup\_py = load\_setup\_py\_data() %}. When the package is built metadata like the version number and license will be read directly from the setup.py file in the source path.

Read the [meta.yaml documentation](https://conda.io/docs/user-guide/tasks/build-packages/define-metadata.html) for more details.

Why are there build: and run: sections in requirements:?

##### Instructions

50 XP

##### Instructions

50 XP

##### Possible Answers

* These two sections define the same package versions.
* Build packages are those required by setup.py. **(A)**
* The build section is intended to be left empty.
* Run requirements are determined from \_\_ini\_\_.py.

# Conda package dependencies

In the meta.yaml file dependent packages and versions are defined using comparison operators, such as <, <=, >, >=. Multiple version conditions are separated by commas. The glob \* means that the preceding characters must match exactly.

Here's an example for a new package called my\_package.

requirements:

run:

- python

- scipy

- numpy 1.11\*

- matplotlib >=1.5, <2.0

Which of the following statements below is INCORRECT?

##### Answer the question

50 XP

##### Possible Answers

* NumPy version 1.13 is compatible with my\_package. **(A)**

press

* Any version of Scipy can be installed with my\_package.

press

* Conda will downgrade matplotlib if version 2 is installed.

press

* my\_package is compatible with Python 2 and Python 3.

press

Correct! Any version of NumPy after 1.11.x or before 1.12.0 can be installed.

# Dependent package versions

The mortgage\_forecasts package you wrote in the last chapter has been provided in your home directory.

In this exercise you'll specify the dependent packages along with their compatible versions.

This project is valid for both Python 2 and Python 3 and depends on the following packages

* Python: any version greater than 2.7
* Pandas: at least version 0.20
* Scipy: any version compatible with Pandas 0.20
* Statsmodels: any version compatible with the above restrictions

##### Instructions 1/2

50 XP

* [1](javascript:void(0))

Edit the meta.yaml file to add a nested run: key under requirements: on line 13 like this

requirements:

run:

- python >=2.7

- pandas >=0.20

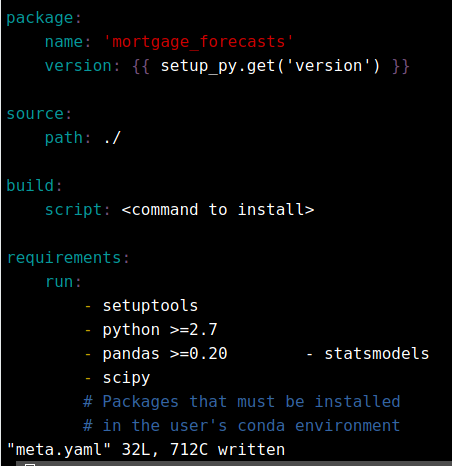
- statsmodels

- scipy

sed -i -e 's/^ run:/ run:\n - python >=2.7\n - pandas >=0.20 - statsmodels\n - scipy/' /home/repl/mortgage\_forecasts/meta.yaml

The next step is to add the build requirements. The packages listed under build: are only those required to run python setup.py. In our case only python and setuptools are required.

At line 20 you'll see the build requirements section. Add python and setuptools to the requirements like you did for the run section.



# Build the Conda Package

Now that you've got the dependent packages defined in your Conda recipe you're ready to build the package.

The complete mortgage\_forecasts Conda package has been loaded into your home directory. Included in the meta.yaml file is the recommended install command for Python-only Conda packages:

build:

script: python setup.py install --single-version-externally-managed --record=record.txt

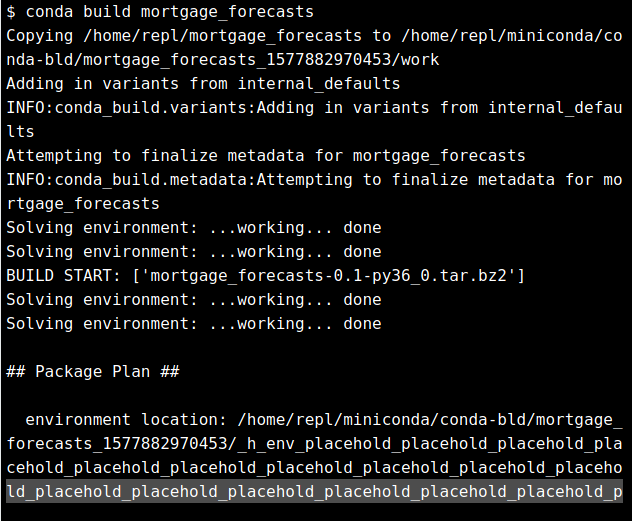
If the script: tag is not present in the meta.yaml file then conda build will expect the installation commands to be in a file called build.sh on Mac and Linux, or in build.bat on Windows.

##### Instructions 1/2

50 XP

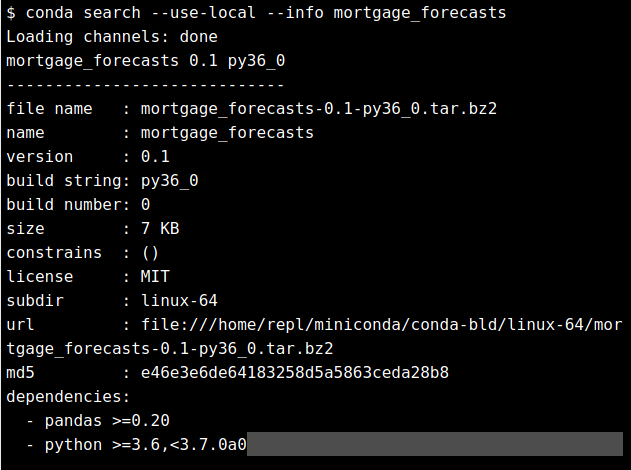
* [1](javascript:void(0))

Run conda build <package-name> with the name of your package.



Now that we have the archive let's verify the metadata using conda search.

Run conda search --use-local --info <package-name> using the name of your package. --use-local instructs Conda to search for packages that were built locally. Remember to replace <package-name>.



# Install the conda package

The mortgage\_forecasts package you built in the last exercise is stored locally at the following path.

/home/repl/miniconda/conda-bld/linux-64/mortgage\_forecasts-0.1-py36\_0.tar.bz2

To install the package you'll use the --use-local flag to include the conda-bld directory when conda install searches for required packages.

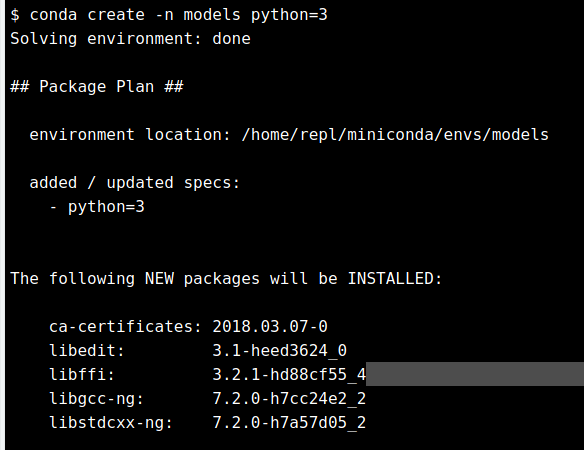
To test that your package is working correctly, you'll make a new Conda environment.

##### Instructions 1/4

25 XP

* [1](javascript:void(0))

Create a new Conda environment called models with only the python=3 package.



Activate the models Conda environment.

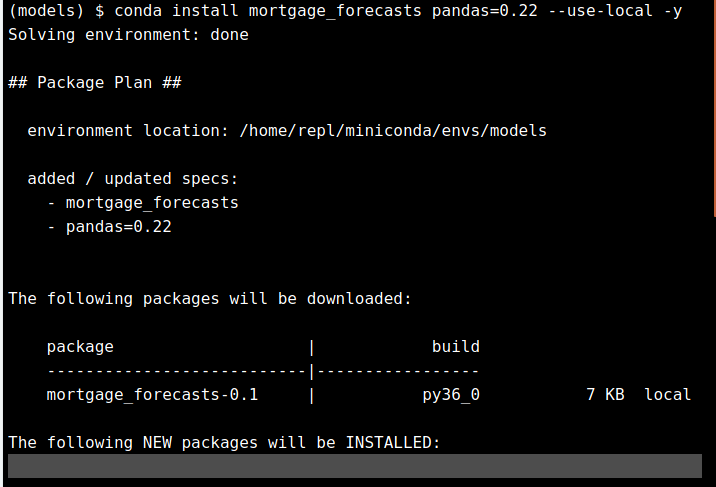


Attempt to install mortgage\_forecasts with pandas version 0.19.

Remember to use --use-local.

conda install mortgage\_forecasts pandas=0.19 --use-local

Now that you've verified that pandas 0.19 cannot be installed with mortgage\_forecasts try again using version 0.22.



# Python versions and architectures

So far you've been developing and building your mortgage\_forecasts package on a 64-bit Linux virtual machine with Python 3. The package you built in the last exercise can only be installed on 64-bit Linux machines along with Python 3.6.

The general practice is to run conda build separately on Windows, Mac, and Linux and with the popular minor revisions of Python (2.7, 3.5, 3.6, etc.). This step is necessary if you have architecture-specific build steps, like compiling [Python C/C++ extensions](https://docs.python.org/3/extending/building.html), or incompatibility between Python 2 and Python 3 in both the build and run steps.

Since your mortgage\_forecasts project is cross-architecture and it works on Python 2 and Python 3 you'll convert it to a noarch Conda package.

This means that only one package archive is required and can be installed with any version of Python on any architecture supported by Conda.

##### Instructions 1/2

50 XP

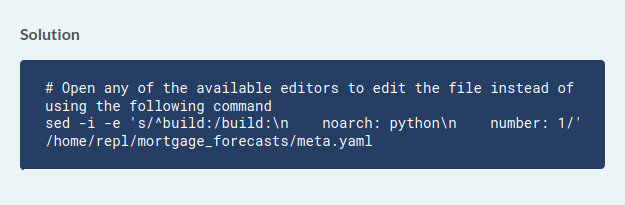
* [1](javascript:void(0))

Edit meta.yaml to add the following line to the build: tag and before the script: tag.

noarch: python

number: 1

By default the build number is 0, so by setting it to 1 we indicate that the Conda package is different from build 0, but the source has not changed. The mortage\_forecasts source is still version 0.1.



Run conda build with the current directory . as the argument.

**ANS : conda build *home/*repl/mortage\_forecasts**

# Upload the package

In the last exercise you created an architecture independent package archive for the mortgage\_forecasts project. The final step is to upload the package to [Anaconda Cloud](https://anaconda.org/) so that others can install it from your channel.

You should have already created an account on [Anaconda Cloud](https://anaconda.org/). You will be asked to enter your login credentials during this exercise.

Your working directory has been set to the conda-bld directory where conda build places package archives.

##### Instructions

100 XP

Run anaconda upload with the following file path as input

noarch/mortgage\_forecasts-0.1-py\_1.tar.bz2

and enter your login and password when prompted.

