TOOLKIT

BTT Cheat Sheets

Table of Contents

TOOL

The Structure of a Notebook

Each body of text or code in a Jupyter Notebook is called a cell. A cell has three possible types:

1. Code
2. Markdown
3. Raw NBConvert

Heading is also listed but has been deprecated in favor of Markdown.

To add a cell — Simply select an existing cell (new notebooks are initialized with one) and press the A key to add a new cell above or the B key to add a new cell below the selected cell.

Alternatively — Select **Insert** in the top menu, and choose **Insert Cell Below** or **Insert Cell Above**.

Running Code

At any given time, one cell is active and awaiting input. The cell that is currently awaiting input is highlighted by a surrounding box and colored bar on the left edge. To execute Python in a cell, the cell type must be set to code. You can execute the cell by either pressing shift-enter or selecting **Run** in the menu bar.

Do **not** use the **Validate** button.

Upon executing the cell, the output of the code, if any, will appear below the cell containing the code. You should execute code cells from top to bottom, in that order, so that the Notebook reads like an annotated program. Prolonged inactivity or navigating to a different page will cause the Notebook session to time out. To resume your work where you left off, be sure to re-run all of the notebook cells in order, starting again at the very beginning.

Note: If you accidentally enter into a markdown cell's editing view, pressing shift-enter or selecting **Run** in the menu bar will return the cell to the rendered markdown view.

Resetting Code

To clear the output, in the menu bar click **Kernel** and select **Restart & Clear Output**. This not only clears all output from executed code but also restarts the kernel, so any set variables or any other persistent action from previously executed code will be lost.

Graded Assignments

Some code cells will be graded. These graded cells will be labeled accordingly. These cells will contain the line # YOUR CODE HERE, and auto-graded cells will contain the line `raise NotImplementedError()`. Remove these lines and replace them with your code. Some of these code cells will be followed by self-check cells that will allow you to test your code before submitting your work for grading. You can run these self-checks as many times as is necessary. Do not write any code in these cells.

Submitting Code

Once you have completed your work on this Notebook, you will submit your code for grading. Follow these steps:

1	**Save your Notebook**	Save your work by selecting the "Save and Checkpoint" entry from the "File" menu at the top of the Notebook.
2	**Mark as Completed**	In the blue menu bar along the top of this code exercise window, you will see a menu item called **Education**. In the **Education** menu, click **Mark as Completed** to submit your code for facilitator review.

Please note: Some notebooks will not contain graded exercises. Therefore, these notebooks will not contain a **Mark as Completed** option.

Downloading Your Jupyter Notebook

If you would like to download a copy of your completed Jupyter Notebook, select **File->Download as** from the menu at the top of the Notebook, and then select your preferred file format. Note: not all formats are supported.

If you choose to download your Notebook in **.ipynb** format, you will notice that once downloaded, your Notebook might have a filename extension **.ipynb.json.** If that is the case, you should change the file extension to **.ipynb** in order to properly open your Notebook using Jupyter on your local machine.

If you would like to produce a static version of the Notebook rendered in HTML, which you might want to share with someone who can view it in a web browser, selection **File->Download as->HTML (.html).**

TOOL

Cheat Sheet: Linux and IPython

Help and Autocompletion

Command	Description
`obj?`	print documentation about obj
`obj??`	print documentation about obj and any available source code defining i
`obj.`	list the attributes of obj (accessible via . typed after obj)
`sometext<AB>`	show list of possible completions after any initial characters

IPython Magic Functions (prefaced with %)

Information on magic functions

Command	Description
`%magic`	print in-depth information about set of available magic functions
`%lsmagic`	print names of all available magic functions
`%timeit?`, `%lsmagic?`, `%who?`	print help for any magic function by appending ? to the end, e.g.,
`%timeit?`	to get information about %timeit

Running code, and inspecting commands and variables

Command	Description
`%run filename`	execute the python code in filename, and bring everything from that module's namespace into the current interactive namespace
`%hist`	print list of full command history in current session • %hist -f filename : save list of full command history to filena
`%who`	print list of all defined variables
`%whos`	print list of all defined variables and their values
`%debug`	activate the interactive debugger after an error has occurred (requires knowledge of Python pdb debugger operation)
`%pprint`	toggle "pretty printing" on and off (may help with viewing contents of large objects)
`%timeit expression`	run specified Python expression, and return information how long it took to run

Interactions with the Operating System (files, etc.)

The commands below are Linux commands that you can run in iPython.

Command	Description
`%ls`	list files in current director
`%more filename`	print the contents of filename to the screen
`%cp`	copy one file to another, or to another directory (e.g., %cp file1 file
`%mv`	rename one file to another, or to another directory (e.g., %mv file1 file
`%mkdir directory_name`	make a new directory called directory_name
`%cd directory_name`	change current directory to specified directory_nam • %pwd : print the current directory
`![command]`	execute the specified command in the system shell (e.g., !ls -l

TOOL

ML Python Packages

Python Standard Library

Where	https://docs.python.org/3/library/index.html
What	A diverse set of modules included in all Python distributions providing a wide range of functions and the ability to interact with the operating system.
When	When you need to manipulate files on your computer, perform some mathematical operations, read files in some standard formats, process text strings that you have read in from a file, figure out where your program is spending most of its time whe it runs... the list goes on.

IPython

Where	ipython.org
What	An extension of the default Python interpreter that provides powerful interactive capabilities to enhance your productivity.
When	When you need to interactively explore the data with which you are working or develop new data science pipelines by figuring out the tools you need and how they can best work with each other.

Jupyter

Where	jupyter.org
What	A system that enables you to create and share documents integrating code, documentation, commentary, results, and data visualizations.
When	When you want to put together a coherent and reproducible analysis documenting a problem on which you are working, its solution, and the results of your analyses.

Matplotlib

Where	matplotlib.org
What	A 2D plotting package that produces publication-quality figures in a variety of formats, with a high degree of customization.
When	When you need to make plots of your data, including line plots, scatter plots, bar charts, heatmaps, histograms, etc.

NumPy

Where	numpy.org
What	The fundamental package for scientific computing in Python, providing multidimensional arrays and a variety of functions for acting on the data stored in arrays.
When	When you want powerful number-crunching capabilities to work with array data, or as part of the larger Python data science ecosystem, which builds new operations on top of the core functionality provided by NumPy.

Pandas

Where	pandas.pydata.org
What	A fast, powerful, flexible, and easy-to-use data analysis and manipulation tool, especially suited to working with tabular data such as is contained in spreadsheets.
When	When you need to read in data from an Excel spreadsheet or CSV file and then process it further by writing your own analysis code.

Scikit-learn

Where	scikit-learn.org
What	A powerful and comprehensive package for doing machine learning in Python, with support for many algorithms as well as testing and cross-validation procedures.
When	When you want to build predictive models from data, such as classification or regression pipelines (supervised learning) or clustering and dimensionality reduction analyses (unsupervised learning)

SciPy

Where	scipy.org
What	An integrated set of packages for computational mathematics, science, and engineering, providing convenient Python interfaces to many well-established algorithms for scientific computin
When	When you want to find the roots of an equation, identify the parameters at the minimum value of a function, integrate a differential equation, fit your data to model, do some signal or image processing, interpolate between two data points… the list goes on.

Seaborn

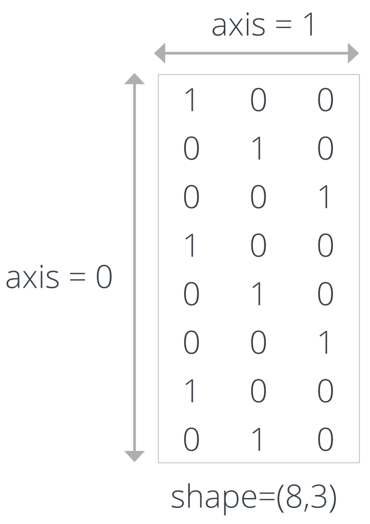
Where	seaborn.pydata.org
What	A data visualization package based on Matplotlib that provides a high-level interface for characterizing statistical properties of data.
When	When you need to visualize structure in your data set, overlaying different components of data into informative visual summaries, with the ability for customization through Matplotlib.

TOOL

NumPy Array Tip Sheet

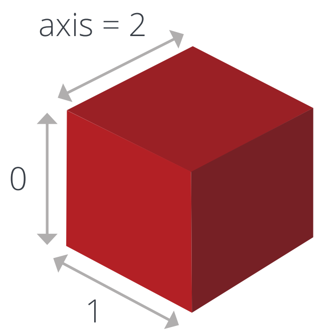
Arrays are the central data type introduced in the NumPy package. Technically, array objects are of type `numpy.ndarray`, which stands for "n-dimensional array." Arrays are accessible by importing the NumPy module, although we will use the conventional shorthand here: `import numpy as np`. Arrays are similar in some respects to Python lists but are multidimensional, homogeneous in type, and support compact and efficient array-level manipulations. Documentation can be found online a www.numpy.org/doc.

Anatomy of an array



The **axes** of an array describe the order of indexing into the array; e.g., axis=0 refers to the first index coordinate, axis=1 the second, etc.

The **shape** of an array is a tuple indicating the number of elements along each axis. An existing array **a** has an attribute **a.shape** which contains this tuple.



- All elements must be the same dtype (data type).
- The default dtype is float
- Arrays constructed from a list of mixed dtype will be upcast to the "greatest" common type.

Constructing arrays

- `np.array(alist)`: Construct an n-dimensional array from a Python list (all elements of list must be of same length).

```
a = np.array([[1,2,3],[4,5,6]])
b = np.array([i*i for i in range(100) if i%2==1])
c = b.tolist()                                   # convert array back to Python list
```

- `np.zeros(shape, dtype=float)`: Construct an n-dimensional array of the specified shape, filled with zeros of the specified dtype.

```
a = np.zeros(100)                    # a 100-element array of float zeros
b = np.zeros((2,8), int)             # a 2x8 array of int zeros
c = np.zeros((N,M,L), bool)          # a NxMxL array of bool zeros
```

- `np.ones(shape, dtype=float)`: Construct an n-dimensional array of the specified shape, filled with ones of the specified dtype.

```
a = np.ones(10, int)                 # a 10-element array of int ones
b = np.pi * np.ones((5,5))           # a useful way to fill up an array with a specified value
```

- `np.transpose(a)`

```
b = np.transpose(a)                  # return new array with a's dimensions reversed
b = a.T                              # equivalent to np.transpose(a)
```

- `np.arange` and `np.linspace`

```
a = np.arange(start, stop, increment)          # like Python range, but with (potentially) real-valued
                                                 arrays
b = np.linspace(start, stop, num_elements)     # create array of equally spaced points based on
                                                 specifed number of points
```

- Random array constructors in `np.random`

```
a = np.random.random((100,100))                # 100x100 array of floats uniform on [0.,1.)
b = np.random.randint(0,10, (100,))            # 100 random ints uniform on [0, 10); i.e., not
                                                 including the upper bound 10
c = np.random.standard_normal((5,5,5))         # zero-mean, unit-variance Gaussian random numbers in a
                                                 5x5x5 array
```

Indexing arrays

- Multidimensional indexing

```
elem = a[i,j,k]                                # extract element at position (i,j,k) in array
```

- "Negative" indexing (wrap around the end of the array)

```
last = a[-1]                                   # the last element of the array (returns array if a.ndim
                                                 > 1)
last_elem = a[-1,-1]                           # the lower-right element of an array (returns array if
                                                 a.ndim > 2)
```

- Arrays as indices

```
i = np.array([0,1,2,1])                        # array of indices for the first axis
j = np.array([1,2,3,4])                        # array of indices for the second axis
a[i,j]                                         # return array([a[0,1], a[1,2], a[2,3], a[1,4]]) based on
                                                 i and j above
#
b = np.array([True, False, True, False])
a[b]                                           # return array([a[0], a[2]]) since only b[0] and b[2] are
                                                 True
```

Slicing arrays (extracting subsections)

- Slice a defined subbloc

```
section = a[10:20, 30:40]                      # 10x10 subblock starting at [10,30]
```

- Grab everything up to the beginning/end of the array

```
asection = a[10:, 30:]                         # missing stop index implies until end of array
bsection = b[:10, :30]                         # missing start index implies until start of array
```

- Grab an entire column

```
x = a[:, 0]                                    # get everything in the 0th column (missing start and
y = a[:, 1]                                      stop)
                                               # get everything in the 1st column
```

- Slice off the tail end of an arra

```
tail = a[-10:]                                 # grab the last 10 elements of the array
slab = b[:, -10:]                              # grab a slab of width 10 off the “side” of the array
interior = c[1:-1, 1:-1, 1:-1]                 # slice out everything but the outer shell
```

Element-wise functions on arrays

- Arithmetic operations

```
c = a + b                                 # add a and b element-wise (must be same shape)
d = c + 2                                 # add 2 to every element of c
h = e * f                                 # multiply e and f element-wise (NOT matrix
                                            multiplication)
g = -h                                    # negate every element of h
m = c ** 2                                # compute the square of every element of c
z = w > 0.0                               # return Boolean array indicating which elements are >
                                            0.0
logspace = 10**np.linspace(-6, -1, 50)    # array of 50 equally spaced-in log points between 1.0e-
                                            06 and 1.0e-01
```

- Trigonometric operations

```
y = np.sin(x)                             # sin of every element of x
w = np.cos(2*np.pi*x)                     # cos of 2*pi*x
```

Summation of arrays

- Simple sums

```
s  = np.sum(a)                            # sum all elements in a, returning a scalar (single
                                            number)
s0 = np.sum(a, axis=0)                    # sum elements along specified axis (=0), returning an
                                            array of remaining shape
s1 = np.sum(a, axis=1)                    # sum elements along axis=1; i.e., over rows if a is
                                            2-dimensional
```

- Averaging, etc.

```
m = np.mean(a, axis)                     # compute mean along the specified axis (over entire
                                           array if axis=None)
s = np.std(a, axis)                      # compute standard deviation along the specified axis
                                           (over entire array if axis=None)
```

- Cumulative sums

```
s0 = np.cumsum(a, axis=0)                # cumulatively sum over 0 axis, returning array with
                                           same shape as a
s0 = np.cumsum(a)                        # cumulatively sum over 0 axis, returning 1D array of
                                           length shape[0]*shape[1]*...*shape[dim-1]
```

Some other useful functions and methods

Many of these work both as separate functions (e.g., `np.sum(a)`) as well as array methods (e.g., `a.sum()`).

- `np.any(a)`: Return True if any element of a is True.
- `np.all(a)`: Return True if all elements of a are True.
- `np.concatenate((a1, a2, ...), axis)`: Concatenate tuple of arrays along specified axis
- `np.min(a, axis=None), np.max(a, axis=None)`: Get min/max values of a along specified axi (global min/max if axis=None).
- `np.argmin(a, axis=None), np.argmax(a, axis=None)`: Get indices of min/max of a along specified axi (global min/max if axis=None).
- `np.reshape(a, newshape)`: Reshape a to new shape (must conserve total number of elements).
- `np.histogram, np.histogram2d, np.histogramdd`: 1-dimensional, 2-dimensional, and d-dimensional histograms, respectively.
- `np.round(a, decimals=0)`: Round elements of array a to specified number of decimals
- `np.sign(a)`: Return array of same shape as a, with -1 where a < 0, 0 where a = 0, and +1 where a > 0.
- `np.abs(a)`: Return array of same shape as a, with the absolute value of each corresponding element.
- `np.unique(a)`: Return sorted unique elements of array a.
- `np.where(condition, x, y)`: Return array with same shape as condition, where values from x are inserted in positions where condition is True, and values from y where condition is False.

TOOL

Structure of a DataFrame Object

df.columns: list-like group of column names

	Date	Size	Base Topping #1	Topping #2	Topping #3	Topping #4	Price
0	2018-02-20	L	Cheese	Garlic	NaN	NaN	15.0
1	2018-02-20	M	Cheese	Onions	NaN	NaN	13.5
2	2018-02-20	S	Margherita	NaN	NaN	NaN	10.0
3	2018-02-20	M	Margherita	NaN	NaN	NaN	12.0
4	2018-02-21	L	Bianco	NaN	NaN	NaN	14.5
5	2018-02-21	M	Marinara	Anchovies	NaN	NaN	13.5
6	2018-02-21	L	Cheese	Pepperoni	Peppers	Mushrooms	16.5

df.index: list-like group of row names

df.values: NumPy array of table data

array dtype is generic “object” type if table data is heterogeneous

array dtype is specific (e.g., int, float) if table data is homogeneo

TOOL

Plotting Tip Sheet: Matplotlib, Pandas, and Seaborn

Plotting in Python is supported by many different packages. This tip sheet focuses on the core functionality provided by Matplotlib, as well as the use of Matplotlib by both Pandas and Seaborn to provide convenient interfaces for plotting data from DataFrames and visually characterizing statistical properties of data. Links to documentation for each of these packages include:

- Matplotlib documentation
- Seaborn documentation
- Pandas plotting documentation

Matplotlib

- General information

```
import matplotlib.pyplot as plt   # import the pyplot interface and name it plt
```

- Multiple function calls using `plt` can be made to build up a figure in stages (e.g., plot multiple data sets in the same figure) customize the plot attributes (axis labels, tick marks, etc.).
- At any given time, a particular figure is active, meaning that new `plt` commands are directed to that figure. If one is generatin multiple figures at once, which figure is active can be switched with th `plt.figure` function as described below.
- Some code examples are given below, but many more possibilities exist, so consult the available documentation for more information.

- `plt.figure:` Create a new figure

`plt.figure()`	`# create a new figure`
`plt.figure(figsize=(10,10))`	`# create a new figure with specified size (width, height in inches)`
`plt.figure(2)`	`# either create a new figure numbered 2, or make figure 2 active if it already exists`
`fig, ax = plt.subplots(2,2)`	`# create grid of subplots with specified number of rows and columns (e.g., (2,2))`
`ax[0,0].plot(x, y)`	`# make plot in subplot by indexing into array of axes produced by plt.subplots`

- `plt.plot:` Create a line plot.

`plt.plot(y)`	`# plot data in list or array y; x-axis defaults to range(len(y))`
`plt.plot(x, y)`	`# plot data in y against data in x; requires x and y to have the same length`
`plt.plot(x, y, 'bo')`	`# plot data in y against data in x, using format string to plot blue (b) circles (o)`
`plt.plot(x, y, 'rs-')`	`# plot data in y against data in x, using format string to plot red (r) squares (s) connected by lines (-)`
`plt.plot(a[:,0], a[:,1])`	`# plot data in column 1 of 2-dimensional array a against data in column 0`
`plt.errorbar(x, y, yerr)`	`# similar to plt.plot, but with error bars around data in y as specified in yerr`

- `plt.scatter:` Create a scatter plot.

`plt.scatter(x, y)`	`# scatter plot data y against the data in x`
`plt.scatter(x, y, s=5, c='b')`	`# scatter plot data y against the data in x, with markers of size 5 and color blue ("b")`
`plt.scatter(x, y, marker='^')`	`# scatter plot data y against the data in x, using marker style "^" (triangle pointing up)`

- **`plt.bar / plt.barh:`** Create a bar chart.

`plt.bar(x, height)`	`# vertical bar chart of data in height, with bars positioned according to data in x`
`plt.bar(x, y, color='r')`	`# vertical bar chart of data in height, with bars positioned according to data in x, coloring the bars red ("r")`
`plt.barh(y, width)`	`# horizontal bar chart of data in width, with bars positioned according to data in y`

- **`plt.hist:`** Create a histogram.

`plt.hist(x)`	`# plot a histogram (with vertical bars), putting data from x into default set of bins`
`plt.hist(x, bins=50)`	`# plot a histogram, putting data from x into specified number of bins (e.g., 50)`
`plt.hist(x, bins=range(0,100))`	`# plot a histogram, putting data from x into bins with specified bin edges`

- Customizing figures (providing additional commands to modify figures already generated with a plotting command

`plt.xlim(0, 100)`	`# set the limits of the x-axis to specified range (e.g., (0, 100))`
`plt.ylim(0.1, 0.9)`	`# set the limits of the y-axis to specified range (e.g., (0.1, 0.9))`
`plt.xlabel('year')`	`# set the label of the x-axis to specified text (e.g., "year")`
`plt.ylabel('cost')`	`# set the label of the y-axis to specified text (e.g., "cost")`
`plt.semilogx()`	`# make the x-axis logarithmic`
`plt.semilogy()`	`# make the y-axis logarithmic`
`plt.loglog()`	`# make both axes logarithmic`
`plt.savefig('mydata.png')`	`# save the current active figure to specified file; file format is detected from file suffix (e.g., ".png")`
`plt.tight_layout()`	`# automatically adjust plot parameters, typically to reduce plot margins`

Pandas

- General information

```
import pandas as pd                # import pandas with shorthand pd
```

- Pandas uses Matplotlib to make plots of data in DataFrames and Series by calling plot methods on those objects rather than using the underlying plt interface directly.
- The plt interface can be used directly, however, to further customize plots generated by Pandas.
- The code examples below assume that there exists a DataFrame named 'df.'

- **`df.plot:`** Plot data in a DataFrame.

`df.plot(x='year', y='cost')`	`# make a line plot of the DataFrame column "cost" against the column "year" (assuming they exist)`
`df.plot(x='year', y='cost', kind='bar')`	`# make a bar plot of "cost" against "year"`
`df.plot.bar((x='year', y='cost')`	`# same as previous example: bar plot of "cost" against "year"`
`df.plot(x='year', y='cost', kind='scatter')`	`# make a scatter plot of "cost" against "year"`
`df.plot.scatter(x='year', y='cost')`	`# same as previous example`
`df.plot.hist('cost')`	`# plot a histogram of the data in column "cost"`
`df.plot.box()`	`# make a box plot (box and whisker) of all the columns in the DataFrame`
`df.groupby('category').mean().plot(x='year', y='cost')`	`# make a plot for a DataFrame derived through other operations such as groupby`

Seaborn

- General information

```
import seaborn as sns                    # import seaborn with shorthand sns
```

- Seaborn uses Matplotlib to make plots of data, usually in DataFrames and Series, by calling plot methods on those objects rather than using the underlying plt interface directly.
- The plt interface can be used directly, however, to further customize plots generated by Seaborn.
- Since Seaborn is focused largely on characterizing statistical properties of data, its functionality is broadly divided into subareas for: (1) visualizing statistical relationships; (2) plotting with categorical data; (3) visualizing the distribution of a data set; and (4) visualizing linear relationships.
- Seaborn contains some predefined d ta sets that can be loaded using the `sns.load_dataset` function; for example:

```
tips = sns.load_dataset('tips')
mpg = sns.load_dataset('mpg')
fmri = sns.load_dataset('fmri')
titanic = sns.load_dataset('titanic')
iris = sns.load_dataset('iris')
```

- The code examples below assume that these sample DataFrames have been loaded and are drawn from the material in the Official Seaborn Tutoria

- `sns.relplot:` Plot relationship between variables in a DataFrame.

```
sns.relplot(x='total_bill', y='tip',    # for tips, make a scatter plot between "total_bill" and "tip"
data=tips)                                (scatter plot: default kind)
sns.relplot(x='total_bill', y='tip',    # same as above, but color each point by categorical data in
hue='smoker', data=tips)                  "smoker"
sns.relplot(x='total_bill', y='tip',    # for tips, make a line plot between "total_bill" and "tip"
kind='line', data=tips)                   (kind="line")
sns.relplot(x='timepoint', y='signal',  # for fmri, make a line plot with 95% confidence interval about
kind='line', ci=95, data=fmri)            mean
```

- `sns.catplot:` Plot with categorical data.

`sns.catplot(x='day', y='total_bill', data=tips)`	`# for each category in x ("day"), make scatter plot of data in y ("total_bill"), with jitter`
`sns.catplot(x='day', y='total_bill', kind='swarm', data=tips)`	`# as above, but splay points as in a "beeswarm" to prevent them from overlapping`
`sns.catplot(x='day', y='total_bill', kind='box', data=tips)`	`# for each category in x, make a box plot for data in y`
`sns.catplot(x='sex', y='survived', hue='class', kind='bar', data=titanic)`	`# make a bar chart, coloring each bar by "class" category`
`sns.catplot(x='deck', kind='count', palette='ch:.25', data=titanic)`	`# make "count" plot (i.e., histogram) for categories in x (with a specified color palette)`
`sns.catplot(x='day', y='total_bill', col='smoker', data=tips)`	`# make multiple catplots in a FacetGrid, in columns for each category in col="smoker"`

- `sns.distplot:` Plot a univariate distribution.

`sns.distplot(tips['total_bill'])`	`# plot a histogram, along with a line representing kernel density estimation (kde=True by default)`
`sns.distplot(tips['total_bill'], kde=False, rug=True)`	`# plot a histogram, along with a "rug" plot showing individual values, but no kde`
`sns.distplot(tips['total_bill'], bins=20, rug=True)`	`# change the number of bins`
`sns.distplot(tips['total_bill'], hist=False, rug=True)`	`# no histogram, but kde and rug plot`

- `sns.jointplot:` Plot bivariate distributions.

```
sns.jointplot(x='total_bill',    # make a scatter plot of x and y, with histograms for each variable
y='tip', data=tips)                along each axis
sns.jointplot(x='total_bill',    # make a "hexbin" plot of x and y, with histograms for each variable
y='tip', kind='hex', data=tips)    along each axis
sns.jointplot(x='total_bill',    # make a kde-based contour plot of x and y, with kde plots for each
y='tip', kind='kde', data=tips)    variable along each axis
```

- `sns.pairplot:` Plot pairwise relationships among variables.

```
sns.pairplot(iris)               # make a pairwise scatter plot grid of columns in "iris" data set
sns.pairplot(iris,
hue='species')                   # color each category of "species" by a distinct color
```

- `sns.regplot and sns.lmplot:` Plot regression models among variables.

```
sns.regplot(x='total_bill',         # scatter plot of x and y, along with best-fit linear regression and
y='tip', data=tips)                   95% confidence interval
sns.lmplot(x='total_bill',          # multiple scatter plots and regression lines, for each category in
y='tip', hue='smoker', data=tips)     hue="smoker"
```

CHEAT SHEET

K-Nearest Neighbors

Algorithm Name	KNN
Description	For a given test point, the KNN algorithm identifies the k most similar training points and finds the most common label among them. This label is used as a prediction for the test point.
Applicability	Often competitive in low-dimensional spaces in settings with many classes; used for classification or regression.
Assumptions	“Similar inputs have similar labels”; KNN assumes that the user has a way to compute distances that reflect meaningful dissimilarities.
Underlying Mathematical Principles	• Distance metrics
Additional Details	• Hyperparameter is number of neighbors (k) • Dealing with ties — fall back to smaller k-values • Distance metric used is application specifi
Example	Identify individuals visible in a photo uploaded to a social media account.

CHEAT SHEET

Decision Trees

Algorithm Name	Decision trees
Description	A framework that solves classification and regression problems
Applicability	• Classification and regression problem • Makes no assumptions regarding feature representation and works with both continuous-valued and categorical features
Assumptions	Similar inputs have similar labels.
Underlying Mathematical Principles	Entropy and information gain are the criteria used to select features and split values at each non-leaf node.
Hyperparameters	max_depth (often alternatively maximum number of nodes), maximum samples per leaf, splitting criterion (information gain)
Additional Details	The classifier is represented by a binary tree
Example	Predict whether an individual will default on a loan based on credit score, age, and loan amount.

CHEAT SHEET

Logistic Regression

Algorithm Name	Logistic regression
Description	Logistic regression is a probabilistic linear model. In essence, a logistic regression classifier produces the probabilit $P(y=1\|X)$ via the inverse logit (sigmoid) function.
Applicability	Binary classification problem
Assumptions	None
Underlying Mathematical Principles	• Linear classification mode • Inverse logit function • Log loss function
Additional Details	• You can use gradient descent to find the optimal solution and yo can tune the learning rate. • Logistic regression uses regularization to reduce model complexity. You can tune hyperparameter C to adjust the penalty.
Example	Predict whether a candidate will win an election based on features such as campaign funds, poll results, if the candidate is currently in office or not, et

TOOL

Deploying, Hosting, and Monitoring Your Model

Introduction

Once you have trained and tested your machine learning model, you have to deploy and host your model in a production environment so as to make your model available to stakeholders to help solve their business problems. Deploying a machine learning model can be challenging, but advancements in cloud computing have made the process much easier.

This reference tool outlines the considerations for deploying, hosting, and monitoring machine learning models to ensure their ongoing success. Use it as a guide when you are prepared to deploy your machine learning model.

1. Determine the deployment method.

After the model is fully trained, it can be deployed to make predictions on new data through two primary methods: batch inference (or offline inference) and online inference (also known as real time). Batch deployment processes large volumes of data in batches on a recurring schedule and stores the predictions in a database. This information can be provided to stakeholders when needed. Online inference processes data as it arrives in real time.

The choice between the two methods depends on the specific problem requirements, and both come with their own set of advantages and disadvantages.

Deployment Method	Pros	Cons
Batch inference	• Can deploy more complex models with a larger number of inputs • Requires simpler infrastructure requirements and less computational power compared to online inference • Predictions can be analyzed and processed before being seen by stakeholders	• Can result in higher processing latency • Not suitable for applications that require real-time predictions, therefore predictions may not be available for new data
Online inference	• Provides results in real time and on demand. • Lower processing latency	• Harder to implement • Cannot handle as complex models as batch inference • More computationally demanding compared to batch inference

Key considerations:

- Your latency requirements
- The complexity of your model
- Specific requirements of your use case

Questions to ask:

- How frequently do you require predictions?
- Do you need results based on individual cases or batches of data?
- What amount of computational power is needed to process the inputs?

2. Choose a hosting environment.

When deploying a machine learning model, one important consideration is where to host it. There are two main options — internal hosting or cloud services — and the decision often depends on the specific needs of the project and organization. Below are some key considerations when choosing where to host your model.

Hosting option	Pros	Cons
Internal	• Greater control and security for sensitive data • May be more cost effective for larger companies with existing internal infrastructure	• Can be more costly in resources, time, and maintenance • More difficult to scale
Cloud services	• Easily scalable and flexible, with lower maintenance • More cost effective for smaller companies without existing internal infrastructure	• May not be cost effective on large projects • May not meet strict security requirements

Questions to ask:

- **Infrastructure:** Does your organization have the necessary hardware, network, and security protocols to support internal hosting? If not, consider cloud services.
- **Cost:** How much will it cost to host and deploy the model? Internal hosting may require a significant upfront investment, while web services are often charged based on usage.
- **Scalability:** Will your project grow in terms of data volume and user demand? Choose a host that can easily scale up to meet those needs, such as cloud-based web services.
- **Security:** Does your project have specific, strict security requirements? If so, deploying the model on-premises can provide better control over the infrastructure and data.

Popular cloud services for deploying and hosting machine learning models

Deploying ML models to the cloud is an increasingly common practice, as it provides easy access to necessary computing power, storage, and network resources for handling large data volumes and running complex models. Yet there are also potential downsides to consider when weighing your options, including security concerns, high cost, latency, and dependency on the cloud provider.

- **Amazon SageMaker** is a fully managed machine learning service offered by Amazon Web Services (AWS). It provides a complete platform for building, training, and deploying machine learning models.
- **Google Cloud AI Platform** is a suite of tools and services that help you build, train, and deploy machine learning models on Google Cloud.
- **Microsoft Azure Machine Learning** is a cloud-based machine learning platform that enables you to build, deploy, and manage machine learning models.
- **BentoML** is an open-source platform for deploying, managing, and serving machine learning models. It provides a unified interface for packaging and deploying models as production-ready web services.
- **Kubeflow** is an open-source platform for deploying and managing machine learning workflows on Kubernetes. It provides a unified interface for managing machine learning pipelines.
- **TensorFlow Serving** is a framework for serving machine learning models using TensorFlow. It provides a flexible architecture for deploying models in production.

3. Deploy your model.

Once you've chosen the deployment method and hosting environment that suits your project, the next step is to package the model along with its dependencies into a deployable format such as a container or a bundle. **Containers** are popular because they're predictable, reproducible, and easily modifiable, making them ideal for collaboration among engineers.

Industry-standard tools that specialize in preparing for deployment:

- **Docker** is a popular tool for packaging and deploying applications in containers. It can be used to package machine learning models and their dependencies into a container that can be easily deployed to different environments.
- **Kubernetes** is an open-source container orchestration platform that can be used to manage and scale containerized applications. It can be used to deploy and manage machine learning models packaged in containers.
- **ONNX Runtime** is an open-source runtime engine for deploying models that are compliant with the Open Neural Network Exchange (ONNX) format. It provides high-performance execution of models across different hardware platforms.

Before deploying the packaged model, you should test it to ensure that it performs as expected. This may involve running tests to evaluate the model's accuracy, precision, recall, and other performance metrics. Once you are satisfied with the results of your testing, deploy the packaged model to your target environment that has been determined based on your security, financial, performance, and computational requirements.

Automating deployment

To streamline deployment and testing workflows, some organizations automate the process. Automation ensures the model is tested regularly to maintain its robustness. It can also help scale the model without burdening the team.

4. Monitor for model improvements.

Monitoring a deployed machine learning model is vital to ensure it is performing well and to detect any issues that may arise during production. Monitoring the model's performance, identifying errors, and making necessary adjustments is crucial. Things to monitor for include:

- **Performance deterioration:** Quality degrades over time due to changes in data distribution.
- **Bias or discrimination:** This occurs when the data used to train the model is not representative of the population it is intended to serve.
- **Security risks:** Over time, attackers may be able to manipulate or alter the model's predictions to achieve their goals.
- **Costly revisions:** A model may require costly revisions or even replacement if its performance degrades over time.

Best practices for monitoring model performance

- Constantly evaluate your ML model's performance on real-world data to detect any decrease in accuracy due to changes in the data environment, known as model drift. If model drift occurs, retrain your model with fresh data to improve its accuracy.
- Monitor your deployment pipeline to ensure the model runs smoothly and debug it when necessary.
- Collect feedback from end users to improve the model's performance by identifying areas for improvement and providing valuable insights.

Considering all of the above factors and being methodical when deploying a machine learning model is important to ensure that the model is accurate, reliable, and delivers the expected value to the business or project for which it is intended.

TOOL

Upload Jupyter Notebooks to GitHub Repository

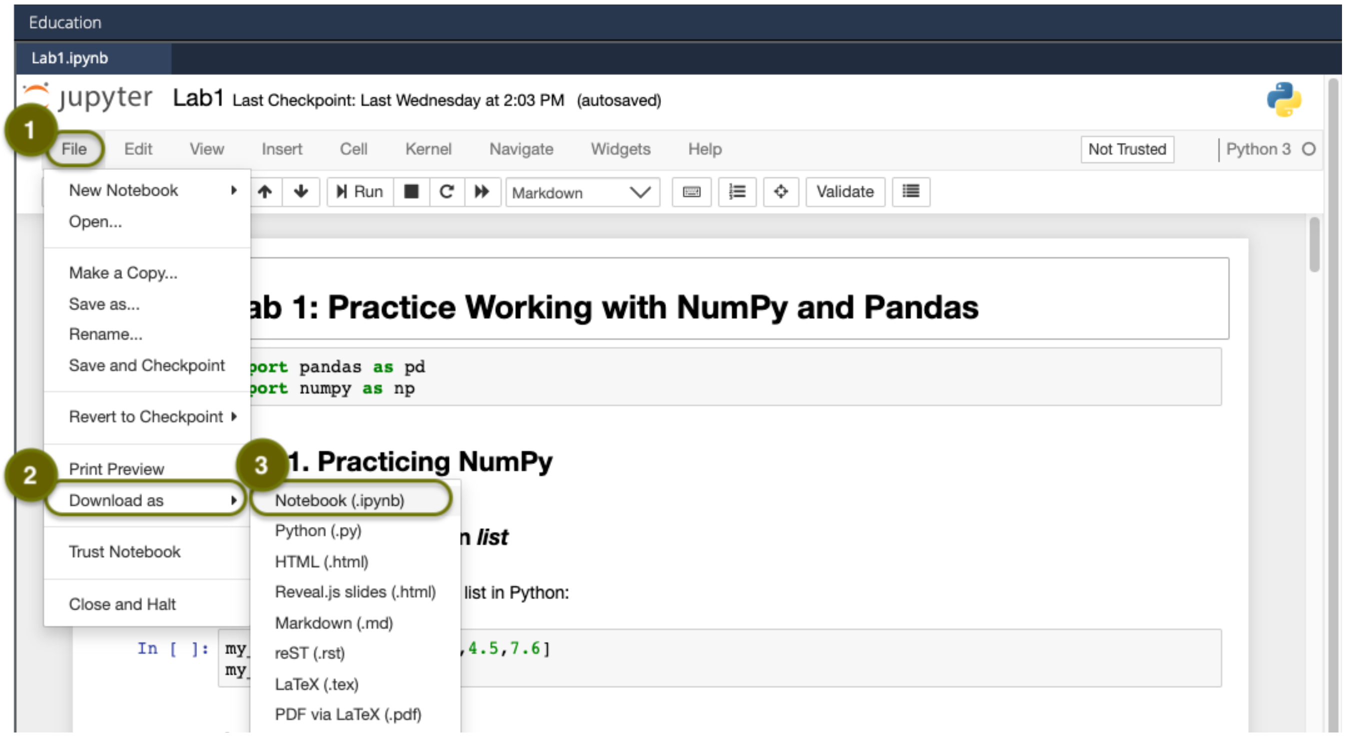
Overview

You may have reason to store and view your Jupyter Notebook assignment files (.ipynb) outside the eCornell Codio environment. Perhaps you want to create a portfolio or revisit assignments after course completion. This can be accomplished using GitHub, a code hosting platform for version control and collaboration. GitHub is a free tool that provides code hosting in a private online repository (A directory or storage space where your projects can live.)

Use the following steps to learn how to download assignment files and then upload them to your own private code repository.

Download the Codio Assignment File

1 In Canvas, in the Jupyter Notebook assignment toolbar, Click File > Download as > Notebook(.ipynb)



```
port pandas as pd
port numpy as np
```

```
In [ ]: my_                ,4.5,7.6]
        my_
```

2 **Optional:** Once downloaded, rename the file on your PC if you want a specific filename prior to uploading to GitHub.

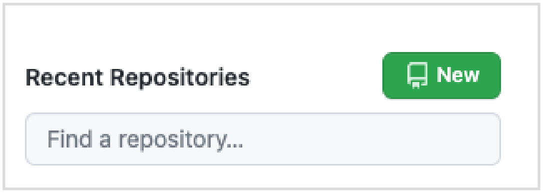
3 **Note:** If you want to capture the notebook's output, you should run it, save it after the process finishes, and download the file.

Create a New Repository on GitHub

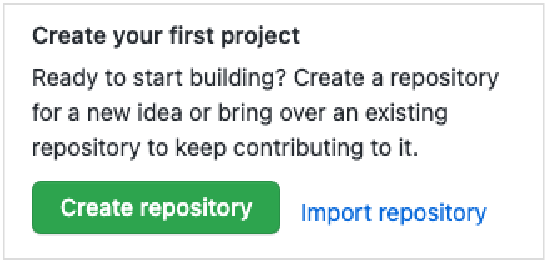
1 If you already have a personal GitHub account, sign in to github.com.

   a. If needed, you can sign up for a free personal GitHub account at github.com. Click **Sign Up** in the upper right corner of the GitHub home page.

2 On the GitHub home page, click **New.**



3 Or click **Create Repository** if you're creating your first repository with a new GitHub account.


Create your first project

Ready to start building? Create a repository for a new idea or bring over an existing repository to keep contributing to it.

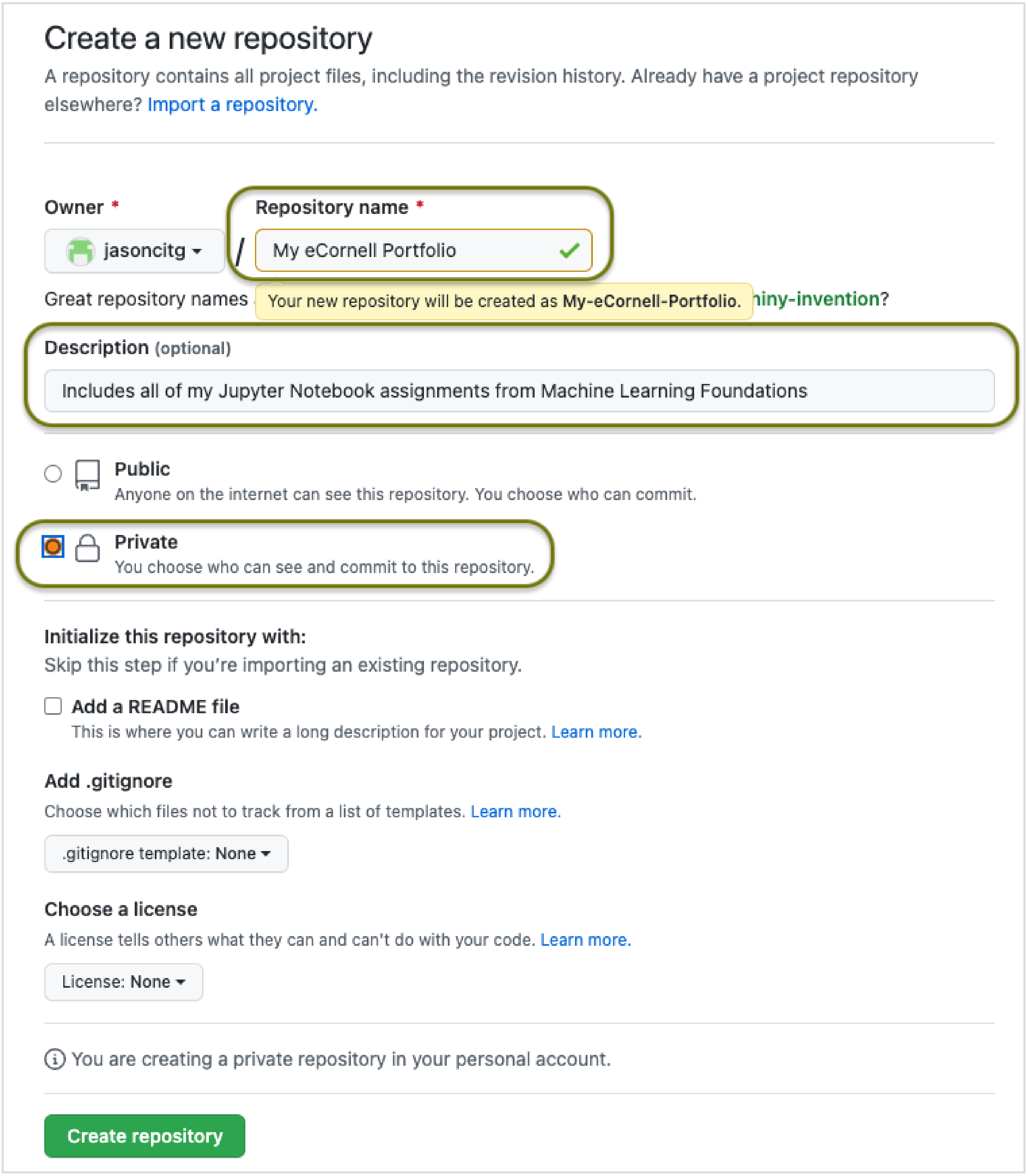
4 Enter a name for the new repository in the **Repository name** field. (ex. My Cornell Portfolio)

5 **Optional:** Enter a description for the portfolio in the description field.

Important: You must set the repository to Private. Not setting it to private could violate Cornell University's Code of Academic Integrity. Before posting, please review Cornell's Code of Academic Integrity and eCornell's policy regarding plagiarism (the presentation of someone else's work as your own without source credit). Failure to set repositories to private could violate this Code and subject you to proceedings under the Code.

6 **Optional:** Check the **Add a README** file box if you wish to write a longer description or include file notes.

7 Click **Create repository**.



Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? Import a repository.

Owner * jasoncitg / **Repository name** * My eCornell Portfolio ✓

Great repository names … Your new repository will be created as **My-eCornell-Portfolio**. …hiny-invention?

Description (optional)

Includes all of my Jupyter Notebook assignments from Machine Learning Foundations

○ **Public**
Anyone on the internet can see this repository. You choose who can commit.

● **Private**
You choose who can see and commit to this repository.

Initialize this repository with:
Skip this step if you're importing an existing repository.

☐ **Add a README file**
This is where you can write a long description for your project. Learn more.

Add .gitignore
Choose which files not to track from a list of templates. Learn more.

.gitignore template: None

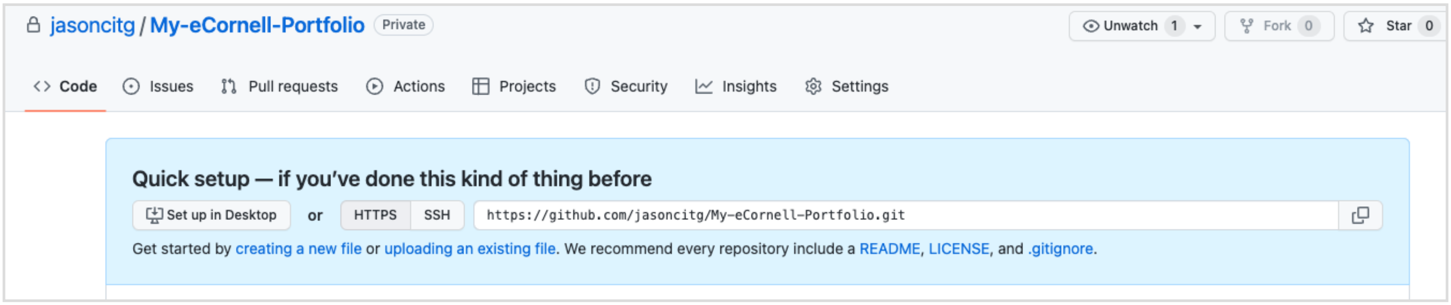
Choose a license
A license tells others what they can and can't do with your code. Learn more.

License: None

ⓘ You are creating a private repository in your personal account.

Upload the Jupyter Notebook File (.ipynb)

GitHub will open the repository page immediately after the repository is created. It's time to upload your first Jupyter Notebook file.



Quick setup — if you've done this kind of thing before

Set up in Desktop or HTTPS SSH https://github.com/jasoncitg/My-eCornell-Portfolio.git

Get started by creating a new file or uploading an existing file. We recommend every repository include a README, LICENSE, and .gitignore.

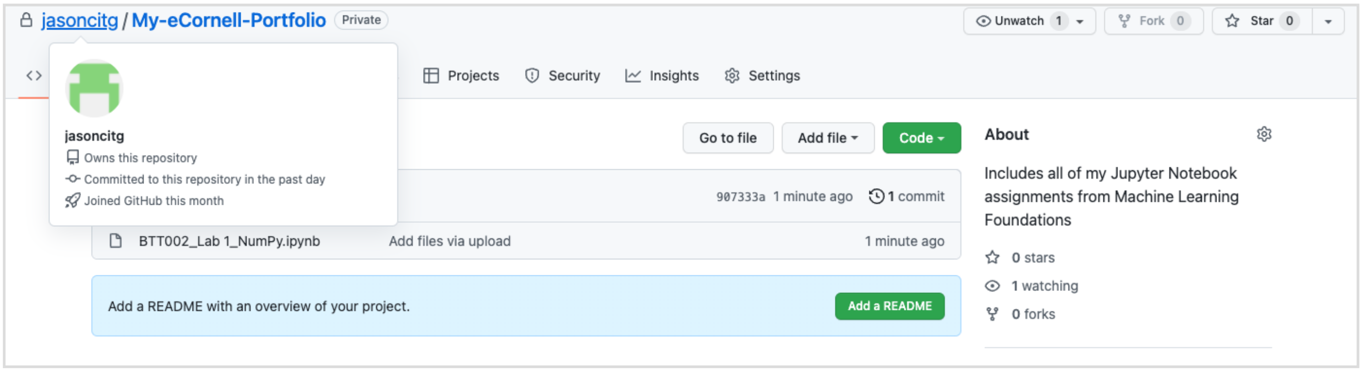
Note: These instructions are for users of the GitHub file upload GUI interface. Please note there are multiple ways to upload a file to GitHub depending on which version (ex., desktop app vs web version.) Visit the GitHub support site to learn about additional file upload methods.

If uploading to a new repository:

1. Click **uploading an existing file** in the top blue section of the upload page.
2. Browse through your directory, click the file to select it (example filename: Lab1.ipynb), and click Open.

If uploading to an existing repository:

1. Open the desired repository, Click **Add file** -> **Upload files**.
2. Click Commit changes. You should receive an upload confirmation on the next page. Please see the example below.



907333a 1 minute ago 1 commit

BTT002_Lab 1_NumPy.ipynb Add files via upload 1 minute ago

Add a README with an overview of your project.

Includes all of my Jupyter Notebook assignments from Machine Learning Foundations

View the Uploaded Jupyter Notebook File

Click the uploaded file link in the main files section of the repository. The code contained in the uploaded file can be viewed on Github in both source and rendered blobs.

- Click on the desired filename from the Code tab.
- Toggle between source and rendered blobs with the small icons in the GitHub file toolbar.

Note: To execute the file, you must download it to a local machine to run via a terminal window or server-client application (ex., Jupyter Notebook app.)

Additional Resources

If you're new to GitHub, it may be helpful to learn more about Git and how it differs from GitHub.

Git	GitHub
Git is a version control system that must be installed on your computer. It is software.	Git is a version control system that must be installed on your computer. It is software.
Ideal for solo users.	Ideal for solo users.
Git is a command-line tool.	Git is a command-line tool.

The following common commands will help you get started in Git:

git init	Turns a directory into an empty Git repository.
git add	Adds files into a staging area for Git.
git commit	Record the changes made to a file to a local repository.
git status	Returns the current state of the selected repository.
git config	Allows the user to assign settings and configurations.
git branch	Determine what branch the local repository is on, add a new branch, or delete a branch.
get checkout	Switch branches
git merge	Integrate branches
git remote	Connect a local repository with a remote repository.
git clone	Create a local working copy of an existing remote repository
git pull	Get the latest version of a repository.
git push	Sends local commits to the remote repository.
git stash	Save changes made when they're not in a state to commit them to a repository.
git log	Show the chronological commit history for a repository.

Refer to this Git Cheat Sheet for additional terminal commands.

CHEAT SHEET

Model Debugging

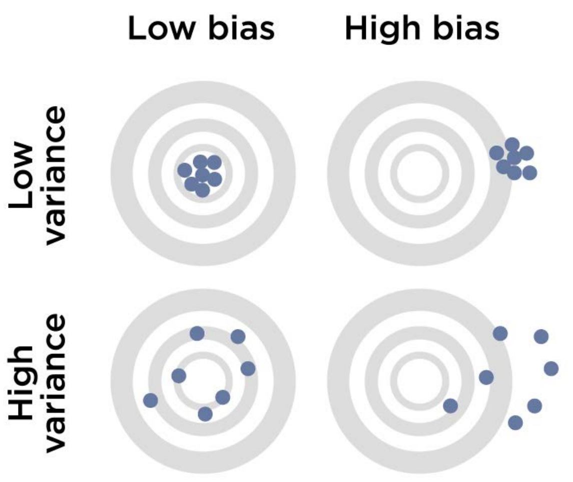
If a model is not performing well, there are several ways to improve its performance. To determine which of the many techniques to use, the first step is to identify the root of the problem.

	High Variance	High Bias
Description	Models with high variance are capable of memorizing many more properties of the training data and do not do well on unseen data, resulting in low training error but high test error.	Models with high bias are too simplistic or make ill-suited assumptions and therefore cannot even achieve low error on the training data set.
Symptoms	Training error is much lower than test error.	Training error is higher than a desired error threshold.
Remedies	• Add more training data • Reduce model complexity (complex models are prone to high variance) • Bagging	• Use a more complex model (or use nonlinear models) • Add features • Boosting

Visualize Variance and Bias

The graph below illustrates data with high/low variance and high/low bias as “darts” thrown at a target. The bullseye at the center of the target is the location of the perfect classifier on the testing data. The blue dots represent the darts, which represent classifiers trained on different training data sets

High variance/low bias models perform well when performance is averaged over large data sets. In expectation they are close to the bullseye but can perform very differently on any two particular data sets. We say that these models are overfit; that they have learned to predict meaningless noise patterns in the data.



High bias/low variance settings lead to models that are very similar across different training data sets (the blue dots are close together); however, they are systematically off-target (i.e., they make wrong assumptions).

The worst case is **high bias and high variance**. The goal is to achieve a model with low bias and low variance.

CHEAT SHEET

Random Forest

Algorithm Name	Random forest
Description	Random forest combines decision trees into an ensemble. The models are trained independently (possibly in parallel) on data bootstraps with one small modification: For each split in each tree, a small subset of k features is randomly sampled and all other features are ignored. This procedure reduces the variance of the final model as training trees on random features results in them being uncorrelated further.
Applicability	Classification and regression problem
Assumptions	Data set is not too high-dimensional; similar inputs have similar labels.
Underlying Mathematical Principles	Full-depth trees have high variance. Random forests combine many high-variance models to create a low-variance ensemble.
Hyperparameters	• Number of trees N to average. • Number of features k to sub-sample. Random forests are famously insensitive to hyperparameters. Default choices: N large enough until the predictions converge (N=100, or N =1000) $k = \sqrt{d}$, where d is the dimensionality of the input data
Setting	Regression or classification. Whil **regression** trees return continuous values, we can still use them to solve **classificatio** problems with discrete labels. For example, we can return the sign of the output of the tree.
Out-of-Bag Error	Because each tree is not trained on the full data but only a sub-sample, each tree has its own out-of-bag subset of the training data on which it was not trained. One can estimate the test error of a random forest classifier by computing the classification error of each data point obtained by averaging the predictions of those ensemble members that were **not** trained on this data point.
Ensemble	Combination of many decision trees
Prediction Confidenc	Random forests naturally provide prediction confidences. For example, in classification settings, you can output the percentage of trees that predicted the most common label as a statistic of confidence. In regression settings, you can output the variance of the predictions of all ensemble members.

Feature Importance	Random forests can naturally provide a measure of feature importance. For example, for each feature, compute the average (or total) reduction in loss obtained on the training set through splitting on this particular feature.
Strengths	Random forests are particularly well suited for applications with heterogeneous features. The big advantages of random forests are that they tend to not overfit to the training data, are amazingly insensitive to hyperparameters, work naturally for regression and classification settings, provide feature importances, output prediction confidences, and provide an unbiased estimate of the testing error directly from the training set.
Weakness	Random forests tend to excel in lower-dimensional feature spaces (<1000 dimensions) and are often not well suited for very high and sparse feature spaces. For very large data sets (n in the millions), random forests can become slow to evaluate, as the trees become too large.

CHEAT SHEET

Gradient Boosted Decision Trees (GBDT)

Algorithm Name	GBDT (an acronym for gradient boosted decision trees)
Description	GBDT is an iterative model that progressively refines its predictions by combining multiple decision trees. The first "tree" simply predicts the average of the output variable, then each successive tree is trained on the residuals from the predictions at the previous level.
Applicability	Classification and regression problem
Assumptions	Independent and identically distributed data
Underlying Mathematical Principles	Adds trees to the ensemble, reduces the residual error in each iteration until there is none
Hyperparameters	Number of trees to add, depth of the trees, and learning rate
Setting	Note that while **regression** trees return continuous values, we can still use them to solve classification problems with discrete labels. For example, we can return the sign of the output of the tree.
Loss Function	For regression, squared loss; for classification, log los

CHEAT SHEET

Neural Network Cheat Sheet

Algorithm Name	Neural networks
Description	Neural networks are versatile models. They use piecewise linear functions to approximate decision boundaries (classification) or the label function (regression).
Applicability	Any supervised learning problem (classification or regression
Assumptions	Input features are homogeneous. Label function is smooth.
Underlying Mathematical Principles	Activation function Loss function Forward propagation Backward propagation Stochastic gradient descent

Activation Functions	ReLU = $\sigma(z) = \max(z, 0)$
	Sigmoid = $\sigma(z) = \frac{1}{1+e^{-z}}$
	tanh = $\sigma(z) = \tanh(z)$

Loss Functions	**Regression:** Squared loss **Classification:** Cross entropy loss
Additional Details	• Training is usually done using SGD — a variant of GD that uses gradient computed from a small batch of training examples (sampled in each iteration). • Gradient is computed efficiently using backwards propagatio • Weight decay and dropout are used to regularize the model.

ConvNet Cheat Sheet

Algorithm Name	Convolutional neural networks
Description	A special type of neural network that is well suited for image data.
Applicability	Any supervised learning problem (classification or regression)
Assumptions	The input is a 2D image of pixels (although 1D, 3D, and 4D variants exist).
Underlying Mathematical Principles	Perceptron Convolution layers Pooling layers Batch normalization
Open Source Implementations	PyTorch TensorFlow
Additional Details	• Training is usually done using SGD • Use cross entropy loss for classification and MSE for regressio • Batch normalization is applied to make training easier • Convolution layers exploit local structures in images

CHEAT SHEET

Recurrent Neural Networks

Algorithm Name	Recurrent neural networks
Description	A special type of neural network that is well suited for sequence data
Applicability	Any supervised learning problem (classification or regression) when the input data are sequences
Assumptions	The data is a discrete sequence
Underlying Mathematical Principles	Hidden states
Additional Details	A recurrent neural network keeps track of a hidden state that can be viewed as a summary of the input sequence. In essence, an RNN processes the input sequence in a stage-wise manner, i.e., it takes in a random starting hidden state and combines it with the first item in the sequence to form a new hidden state, and this new hidden state is combined with the next item in the sequence to produce another new hidden state. Sometimes the RNN also produces an output at each state, which is typically a linear transformation of the hidden state at this position. This process is repeated until every item in the sequence is processed. Depending on the task, the final hidden state will be used differently. If we want to classify a sequence (sentiment classification) then the final hidden state will be passed to a classifier. If we want t produce another sequence (machine translation), then the hidden state can be used as the initial hidden state of another RNN.