This is a tutorial on how to do some typical statistical programming tasks using Python. It's intended for people basically familiar with Python and experienced at statistical programming in a language like R, Stata, SAS, SPSS, or MATLAB.

```
# 0. Getting set up ====

""" Get set up with IPython and pip install the following: numpy, scipy, pandas,
    matplotlib, seaborn, requests.
        Make sure to do this tutorial in the IPython notebook so that you get
    the inline plots and easy documentation lookup.
"""

# 1. Data acquisition ====

""" One reason people choose Python over R is that they intend to interact a lot
    with the web, either by scraping pages directly or requesting data through
    an API. You can do those things in R, but in the context of a project
    already using Python, there's a benefit to sticking with one language.
"""

import requests  # for HTTP requests (web scraping, APIs)
import os

# web scraping
r = requests.get("https://github.com/adambard/learnxinyminutes-docs")
r.status_code  # if 200, request was successful
r.text  # raw page source
print(r.text)  # prettily formatted
# save the page source in a file:
os.getcwd()  # check what's the working directory
f = open("learnxinyminutes.html", "wb")
f.write(r.text.encode("UTF-8"))
f.close()

# downloading a csv
fp = "https://raw.githubusercontent.com/adambard/learnxinyminutes-docs/master/"
fn = "pets.csv"
r = requests.get(fp + fn)
print(r.text)
f = open(fn, "wb")
f.write(r.text.encode("UTF-8"))
f.close()

""" for more on the requests module, including APIs, see
    http://docs.python-requests.org/en/latest/user/quickstart/
"""

# 2. Reading a CSV file ====

""" Wes McKinney's pandas package gives you 'DataFrame' objects in Python. If
    you've used R, you will be familiar with the idea of the "data.frame" already.
```

```
"""

import pandas as pd
import numpy as np
import scipy as sp
pets = pd.read_csv(fn)
pets
#        name  age  weight species
# 0    fluffy    3      14     cat
# 1  vesuvius    6      23    fish
# 2       rex    5      34     dog

""" R users: note that Python, like most normal programming languages, starts
    indexing from 0. R is the unusual one for starting from 1.
"""

# two different ways to print out a column
pets.age
pets["age"]

pets.head(2)  # prints first 2 rows
pets.tail(1)  # prints last row

pets.name[1]  # 'vesuvius'
pets.species[0]  # 'cat'
pets["weight"][2]  # 34

# in R, you would expect to get 3 rows doing this, but here you get 2:
pets.age[0:2]
# 0    3
# 1    6

sum(pets.age) * 2  # 28
max(pets.weight) - min(pets.weight)  # 20

""" If you are doing some serious linear algebra and number-crunching, you may
    just want arrays, not DataFrames. DataFrames are ideal for combining columns
    of different types.
"""

# 3. Charts ====

import matplotlib as mpl
import matplotlib.pyplot as plt
%matplotlib inline

# To do data vizualization in Python, use matplotlib

plt.hist(pets.age);

plt.boxplot(pets.weight);

plt.scatter(pets.age, pets.weight)
plt.xlabel("age")
```

```
plt.ylabel("weight");

# seaborn sits atop matplotlib and makes plots prettier

import seaborn as sns

plt.scatter(pets.age, pets.weight)
plt.xlabel("age")
plt.ylabel("weight");

# there are also some seaborn-specific plotting functions
# notice how seaborn automatically labels the x-axis on this barplot
sns.barplot(pets["age"])

# R veterans can still use ggplot
from ggplot import *
ggplot(aes(x="age",y="weight"), data=pets) + geom_point() + labs(title="pets")
# source: https://pypi.python.org/pypi/ggplot

# there's even a d3.js port: https://github.com/mikedewar/d3py

# 4. Simple data cleaning and exploratory analysis ====

""" Here's a more complicated example that demonstrates a basic data
    cleaning workflow leading to the creation of some exploratory plots
    and the running of a linear regression.
        The data set was transcribed from Wikipedia by hand. It contains
    all the Holy Roman Emperors and the important milestones in their lives
    (birth, death, coronation, etc.).
        The goal of the analysis will be to explore whether a relationship
    exists between emperor birth year and emperor lifespan.
    data source: https://en.wikipedia.org/wiki/Holy_Roman_Emperor
"""

# load some data on Holy Roman Emperors
url = "https://raw.githubusercontent.com/e99n09/R-notes/master/data/hre.csv"
r = requests.get(url)
fp = "hre.csv"
f = open(fp, "wb")
f.write(r.text.encode("UTF-8"))
f.close()

hre = pd.read_csv(fp)

hre.head()
"""
   Ix      Dynasty        Name        Birth             Death Election 1
0 NaN  Carolingian   Charles I  2 April 742    28 January 814        NaN
1 NaN  Carolingian     Louis I          778       20 June 840        NaN
2 NaN  Carolingian   Lothair I          795  29 September 855        NaN
3 NaN  Carolingian    Louis II          825     12 August 875        NaN
4 NaN  Carolingian  Charles II  13 June 823     6 October 877        NaN

  Election 2      Coronation 1   Coronation 2 Ceased to be Emperor
```

```
0        NaN   25 December 800            NaN       28 January 814
1        NaN  11 September 813  5 October 816          20 June 840
2        NaN       5 April 823            NaN     29 September 855
3        NaN        Easter 850     18 May 872        12 August 875
4        NaN   29 December 875            NaN        6 October 877

  Descent from whom 1 Descent how 1 Descent from whom 2 Descent how 2
0                 NaN           NaN                 NaN           NaN
1           Charles I           son                 NaN           NaN
2             Louis I           son                 NaN           NaN
3           Lothair I           son                 NaN           NaN
4             Louis I           son                 NaN           NaN
"""

# clean the Birth and Death columns

import re  # module for regular expressions

rx = re.compile(r'\d+$')  # match trailing digits

""" This function applies the regular expression to an input column (here Birth,
    Death), flattens the resulting list, converts it to a Series object, and
    finally converts the type of the Series object from string to integer. For
    more information into what different parts of the code do, see:
      - https://docs.python.org/2/howto/regex.html
      - http://stackoverflow.com/questions/11860476/how-to-unlist-a-python-list
      - http://pandas.pydata.org/pandas-docs/stable/generated/pandas.Series.html
"""

def extractYear(v):
    return(pd.Series(reduce(lambda x, y: x + y, map(rx.findall, v), [])).astype(int))

hre["BirthY"] = extractYear(hre.Birth)
hre["DeathY"] = extractYear(hre.Death)

# make a column telling estimated age
hre["EstAge"] = hre.DeathY.astype(int) - hre.BirthY.astype(int)

# simple scatterplot, no trend line, color represents dynasty
sns.lmplot("BirthY", "EstAge", data=hre, hue="Dynasty", fit_reg=False);

# use scipy to run a linear regression
from scipy import stats
(slope, intercept, rval, pval, stderr) = stats.linregress(hre.BirthY, hre.EstAge)
# code source: http://wiki.scipy.org/Cookbook/LinearRegression

# check the slope
slope  # 0.0057672618839073328

# check the R^2 value:
rval**2  # 0.020363950027333586

# check the p-value
pval  # 0.34971812581498452
```

```
# use seaborn to make a scatterplot and plot the linear regression trend line
sns.lmplot("BirthY", "EstAge", data=hre);

""" For more information on seaborn, see
      - http://web.stanford.edu/~mwaskom/software/seaborn/
      - https://github.com/mwaskom/seaborn
    For more information on SciPy, see
      - http://wiki.scipy.org/SciPy
      - http://wiki.scipy.org/Cookbook/
    To see a version of the Holy Roman Emperors analysis using R, see
      - http://github.com/e99n09/R-notes/blob/master/holy_roman_emperors_dates.R
"""
```

If you want to learn more, get *Python for Data Analysis* by Wes McKinney. It's a superb resource and I used it as a reference when writing this tutorial.

You can also find plenty of interactive IPython tutorials on subjects specific to your interests, like Cam Davidson-Pilon's Probabilistic Programming and Bayesian Methods for Hackers.

Some more modules to research: - text analysis and natural language processing: nltk, http://www.nltk.org - social network analysis: igraph, http://igraph.org/python/