# Data Science With Python

*L A Liggett*

*2019-06-25*

# Contents

# Chapter 1

# Overview

These are some notes that may be helpful for computational biology analysis that focuses on Python use.

# Chapter 2

# Python

## 2.1   General

Here is a general skeleton that can be used to start a python script that takes input.

```python
#!/usr/bin/env python

def runArgparse():
import argparse

parser = argparse.ArgumentParser()
parser.add_argument('--indir', '-i', type=str, nargs='*', help='Input directory containing the vcf files
parser.add_argument('--loadolddata', '-o', action='store_true', help='Load previously existing data.')

args = parser.parse_args()
indir = args.indir
return indir

if __name__ == '__main__':
    runArgparse()
```

## 2.2   Numba

Numba speeds up python code without having to switch to a different interpreter, and doesn't require static typing of variables as Cython does. Just calling Numba will increase the speed of a script (except during the compilation which will add some time). But this isn't the best way to take advantage of the speed boost.

Here is an example script that uses jit to invoke Numba.

```python
#!/usr/bin/env python

from numba import jit
import numpy as np
import time

def go_slow(x):
```

```python
    for i in range(14):
        x *= x

@jit(nopython=True)
def go_fast(x):
    for i in range(14):
        x *= x

# DO NOT REPORT THIS... COMPILATION TIME IS INCLUDED IN THE EXECUTION TIME!
start = time.time()
go_slow(5)
end = time.time()
print("Elapsed slow (with compilation) = %s" % (end - start))
start = time.time()
go_fast(5)
end = time.time()
print("Elapsed fast (with compilation) = %s" % (end - start))

# DO NOT REPORT THIS... COMPILATION TIME IS INCLUDED IN THE EXECUTION TIME!
start = time.time()
go_slow(5)
end = time.time()
print("Elapsed slow (after compilation) = %s" % (end - start))
start = time.time()
go_fast(5)
end = time.time()
print("Elapsed fast (after compilation) = %s" % (end - start))
```

Parallelization can be used to automatically utilize multiple cores.

```python
from numba import njit, prange
@njit(parallel=True)
def prange_test(A):
    s = 0
    for i in prange(A.shape[0]):
        s += A[i]
    return s
```

## 2.3 Cython

Installation

```
conda install -c conda-forge cython
```

Ipython usage

```python
%load_ext Cython

%%cython
def f(x):
    return 2 * x
```

```
or

def f(int x):
    return 2 * x

timeit(f(4))
```

# Chapter 3

# JupyterLab

Here is a simple template that I use that controls a couple useful things when starting a new notebook.

```python
import sys
sys.path.append('../util')

%reload_ext autoreload
%autoreload 2

from util import *
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns

sns.set_palette('pastel')
sns.set_style('ticks')
sns.set_context('paper', font_scale=1)
```

It is often convenient to have a notebook automatically refresh the imported libraries so that they can be modified while working on a JupyterLab notebook.

```python
%reload_ext autoreload
%autoreload 2
```

To allow directory organization, dependcies can be separated into different directories and imported into a jupyter notebook using the following import statement.

```python
import sys
sys.path.append('../util')
```

A table of contents can be created to refer to each of the headers throughout a notebook in html format. The code is below (Obviously needs to be simplified.)

```html
<h1>Table of Contents<span class="tocSkip"></span></h1>
<div class="toc">
    <ul class="toc-item">
```

```
<li>
    <span><a href="#Python-Setup" data-toc-modified-id="Python-Setup-1"><span class="toc-item-num">
    <ul class="toc-item">
<li>
    <span><a href="#Change-the-width-of-the-page" data-toc-modified-id="Change-the-width-of-the-page
    <li>
        <span><a href="#Import-packages" data-toc-modified-id="Import-packages-1.2"><span class="to
    </ul>
</li>
<li>
    <span><a href="#Colours" data-toc-modified-id="Colours-2"><span class="toc-item-num">2 &nb
    <ul class="toc-item">
<li><span><a href="#Colour-line-graph" data-toc-modified-id="Colour-line-graph-2.1"><span class="to
    </ul>
</li>
<li><span><a href="#Totals-for-studies" data-toc-modified-id="Totals-for-studies-3"><span class="to
<li><span><a href="#Functions-for-calculating-trinucleotide-context-specific-mutation-rates" data-t
    <ul class="toc-item">
    <li><span><a href="#Calculating-mutation-rates-for-individual-variants" data-toc-modified-id="Ca
        <ul class="toc-item">
    <li><span><a href="#DNMT3A" data-toc-modified-id="DNMT3A-4.1.1"><span class="toc-item-num">4.1.
        <li><span><a href="#TET2" data-toc-modified-id="TET2-4.1.2"><span class="toc-item-num">4.1.2
        <li><span><a href="#ASXL1" data-toc-modified-id="ASXL1-4.1.3"><span class="toc-item-num">4.1
        <li><span><a href="#TP53" data-toc-modified-id="TP53-4.1.4"><span class="toc-item-num">4.1.4
    <li><span><a href="#Calculating-mutation-rates-from-a-.csv-file-of-variants" data-toc-modified-i
        <ul class="toc-item">
        <li><span><a href="#DNMT3A" data-toc-modified-id="DNMT3A-4.2.1"><span class="toc-item-num">4
        <li><span><a href="#TET2" data-toc-modified-id="TET2-4.2.2"><span class="toc-item-num">4.2.2
        <li><span><a href="#ASXL1" data-toc-modified-id="ASXL1-4.2.3"><span class="toc-item-num">4.2
        <li><span><a href="#TP53" data-toc-modified-id="TP53-4.2.4"><span class="toc-item-num">4.2.4
    <li><span><a href="#Calculating-mutation-rates-from-a-list-of-variants" data-toc-modified-id="Ca
        <ul class="toc-item">
        <li><span><a href="#DNMT3A" data-toc-modified-id="DNMT3A-4.3.1"><span class="toc-item-num">4
    <li><span><a href="#TET2" data-toc-modified-id="TET2-4.3.2"><span class="toc-item-num">4.3.2&nbs
        <li><span><a href="#ASXL1" data-toc-modified-id="ASXL1-4.3.3"><span class="toc-item-num">4.3
        <li><span><a href="#TP53" data-toc-modified-id="TP53-4.3.4"><span class="toc-item-num">4.3.4
<li><span><a href="#Lists-of-variants-targeted-by-each-study" data-toc-modified-id="Lists-of-variant
    <li><span><a href="#Genovese-2014" data-toc-modified-id="Genovese-2014-5.2"><span class="toc-ite
    <li><span><a href="#McKerrel-2015" data-toc-modified-id="McKerrel-2015-5.3"><span class="toc-ite
    <li><span><a href="#Zink-2017" data-toc-modified-id="Zink-2017-5.4"><span class="toc-item-num">5
    <li><span><a href="#Coombs-2017" data-toc-modified-id="Coombs-2017-5.5"><span class="toc-item-nu
    <li><span><a href="#Young-2016-&amp;-2019" data-toc-modified-id="Young-2016-&amp;-2019-5.6"><spa
    <li><span><a href="#Desai-2018" data-toc-modified-id="Desai-2018-5.7"><span class="toc-item-num
    <li><span><a href="#Acuna-Hidalgo-2017" data-toc-modified-id="Acuna-Hidalgo-2017-5.8"><span clas
<li><span><a href="#Lists-of-all-possible-variants-in-DNMT3A,-TET2,-ASXL1,-TP53" data-toc-modified-
    <ul class="toc-item">
    <li><span><a href="#DNMT3A" data-toc-modified-id="DNMT3A-6.1"><span class="toc-item-num">6.1&nb
    <li><span><a href="#TET2" data-toc-modified-id="TET2-6.2"><span class="toc-item-num">6.2 &n
    <li><span><a href="#ASXL1" data-toc-modified-id="ASXL1-6.3"><span class="toc-item-num">6.3&nbsp
<li><span><a href="#TP53" data-toc-modified-id="TP53-6.4"><span class="toc-item-num">6.4 &nbsp
<li><span><a href="#Actual-number-of-observations-of-each-variant" data-toc-modified-id="Actual-num
    <ul class="toc-item">
    <li>
```

```html
            <ul class="toc-item">
            <li><span><a href="#DNMT3A" data-toc-modified-id="DNMT3A-7.0.1"><span class="toc-item-num">7
            <li><span><a href="#TET2" data-toc-modified-id="TET2-7.0.2"><span class="toc-item-num">7.0.2
        <li><span><a href="#ASXL1" data-toc-modified-id="ASXL1-7.0.3"><span class="toc-item-num">7.0.3&n
        <li><span><a href="#TP53" data-toc-modified-id="TP53-7.0.4"><span class="toc-item-num">7.0.4&nbs
    <li><span><a href="#Functions-for-calculating-the-expected-number-of-observations-of-a-variant" data
    <li><span><a href="#Maximum-Likelihood-Estimation-for-s" data-toc-modified-id="Maximum-Likelihood-Es
        <ul class="toc-item"><li><span><a href="#DNMT3A-variants" data-toc-modified-id="DNMT3A-variants-
        <li><span><a href="#TET2-variants" data-toc-modified-id="TET2-variants-9.2"><span class="toc-ite
        <li><span><a href="#ASXL1-variants" data-toc-modified-id="ASXL1-variants-9.3"><span class="toc-i
        <li><span><a href="#TP53-variants" data-toc-modified-id="TP53-variants-9.4"><span class="toc-ite
        </ul>
        </li>
    </ul>
</div>
```

# Chapter 4

# Visualization

## 4.1 Color

### 4.1.1 Colorschemes

Seaborn Themes

```
Pastel: {'Blue':'#a3c6ff', 'Orange':'#f7ab60', 'Green':'#60f7a9', 'Red':'#fc9d94', 'Purple':'#bea3ff',
```

```
Deep: {'Green':'#5baf68'}
```

### 4.1.2 Controlling Coloration

Not all plots automatically plot with a white background, and when using something dark like jupyterlab or a presentation this can be frustrating. The background color can be set in pyplot like this.

```
fig.patch.set_facecolor('xkcd:mint green')
```

When plotting, samples will not always be colored with the same color, especially when different subsets of samples are included in different plots. Here is a manual workaround to specify the coloration of displayed data. This is a bit cumbersome so there might be a more elegant way of achieving the same outcome.

```python
# here is an example where sample order is controlled from a pandas DataFrame
sample_order = all_vars.sort_values(['ID']).drop_duplicates(['Sample']).Sample

# the color order is specified here
# colors should be in the same order as the above sample_order Series, excluding samples with no data
colors = [pastel['Brown'], pastel['Blue'],
         pastel['Orange'], pastel['Purple'],
         pastel['Green'], pastel['Red'],
         ]

plt.figure()
# this is an example of plotting that uses the sample_order and palette to control coloration order
sns.catplot(x='Sample', y='VAF', hue='Gene', jitter=True,
            data=oncogenic[oncogenic.Location == 'Peripheral'],
```

```
            legend=False, order=sample_order, palette=sns.color_palette(colors))

# a colorscheme can be specified if desired
pastel = {'Blue':'#a3c6ff', 'Orange':'#f7ab60',
          'Green':'#60f7a9', 'Red':'#fc9d94',
          'Purple':'#bea3ff', 'Brown':'#d1b485',
          'Pink':'#f7afdf', 'Gray':'#c4c4c4',
          'Yellow':'#ffffaa', 'LBlue':'#baf6ff'}

# this controls the coloration in the legend
import matplotlib.patches as mpatches
egfr = mpatches.Patch(color=pastel['Blue'], label='EGFR')
pik3ca = mpatches.Patch(color=pastel['Orange'], label='PIK3CA')
myc = mpatches.Patch(color=pastel['Green'], label='MYC')

plt.legend(handles=[egfr,pik3ca,myc],
           loc='upper right', bbox_to_anchor=(1.5, 1),
           ncol=1) # no legend overlap
```

## 4.2   Matplotlib

Plotting a heatmap.

```
import matplotlib.pyplot as plt
import numpy as np
a = np.random.random((16, 16))
plt.imshow(a, cmap='RdBu'', interpolation='nearest')
plt.show()
```

Possible heatmap colors are:

```
Accent, Accent_r, Blues, Blues_r, BrBG, BrBG_r, BuGn, BuGn_r, BuPu, BuPu_r, CMRmap, CMRmap_r, Dark2, Da
Set1_r, Set2, Set2_r, Set3, Set3_r, Spectral, Spectral_r, Wistia, Wistia_r, YlGn, YlGnBu, YlGnBu_r, YlG
gist_stern, gist_stern_r, gist_yarg, gist_yarg_r, gnuplot, gnuplot2, gnuplot2_r, gnuplot_r, gray, gray_
twilight, twilight_r, twilight_shifted, twilight_shifted_r, viridis, viridis_r, vlag, vlag_r, winter, w
```

A simple venn diagram.

```
from matplotlib_venn import venn2
venn2(subsets = (3, 2, 1))
```

A more complicated venn diagram.

```
from matplotlib import pyplot as plt
import numpy as np
from matplotlib_venn import venn3, venn3_circles
plt.figure(figsize=(4,4))
v = venn3(subsets=(1, 1, 1, 1, 1, 1, 1), set_labels = ('A', 'B', 'C'))
v.get_patch_by_id('100').set_alpha(1.0)
v.get_patch_by_id('100').set_color('white')
```

```
v.get_label_by_id('100').set_text('Unknown')
v.get_label_by_id('A').set_text('Set "A"')
c = venn3_circles(subsets=(1, 1, 1, 1, 1, 1, 1), linestyle='dotted')
c[0].set_lw(1.0)
c[0].set_ls('dotted')
plt.title("Sample Venn diagram")
plt.annotate('Unknown set', xy=v.get_label_by_id('100').get_position() - np.array([0, 0.05]), xytext=(-
            ha='center', textcoords='offset points', bbox=dict(boxstyle='round,pad=0.5', fc='gray', alp
                            arrowprops=dict(arrowstyle='->', connectionstyle='arc3,rad=0.5',color='gray')]
                            plt.show()
```

An upset plot is a nice alternative to a traditional venn diagram. The project is hosted here, and this is the documentation.

First install the library.

```
pip install upsetplot
```

Here is the code to create the plot.

```
import numpy as np

arrays = [[False,False,False,False,True,True,True,True],
          [False,False,True,True,False,False,True,True],
          [False,True,False,True,False,True,False,True]]
tuples = list(zip(*arrays))

def o(one=False, two=False, three=False):
    if three:
        temp = pd.merge(indels[(indels.Individual==one)], indels[(indels.Individual==two)], how='inner'
        return len(pd.merge(temp, indels[(indels.Individual==three)], how='inner', on=['Loc', 'Var']))
    elif two:
        return len(pd.merge(indels[(indels.Individual==one)], indels[(indels.Individual==two)], how='in
    elif one:
        return len(indels[(indels.Individual==one)])
    else:
        return 0

index = pd.MultiIndex.from_tuples(tuples, names=['Ind 1', 'Ind 2', 'Ind 3'])
s = pd.Series([o(),
               o(3),
               o(2),
               o(2,3),
               o(1),
               o(1,3),
               o(1,2),
               o(1,2,3)], index=index)

from upsetplot import plot as up
up(s)
plt.savefig("../images/indels.svg", format="svg", bbox_inches="tight")
```

Log scales seem to always be a challenge. Here is at least one solution to change ticks to log manually.

```python
y_major_ticks = [np.log(100),np.log(200),np.log(300),np.log(400),np.log(500),np.log(600),np.log(700),np
                np.log(1000),np.log(2000),np.log(3000),np.log(4000),np.log(5000),np.log(6000),np.log(70
                np.log(10000),np.log(20000),np.log(30000),np.log(40000),np.log(50000),np.log(60000),np
                np.log(100000),np.log(200000),np.log(300000),np.log(400000),np.log(500000),np.log(60000
                np.log(1000000),np.log(2000000),np.log(3000000),np.log(4000000),np.log(5000000),np.log
                np.log(10000000)]

y_major_tick_labels = ["100","","","","","","","","", "1000","","","","","","","","", "10,000",\
                    "","","","","","","","","100,000","","","","","","","","", "1,000,000","","","",
ax1.set_yticks(y_major_ticks)
ax1.set_yticklabels(y_major_tick_labels, fontsize = axisfont)
ax1.yaxis.set_tick_params(width=scale, color = grey3, length = 6)
```

## 4.3  Seaborn

Here is a general bar plot that includes some commonly used parameters.

```python
# fits my 22 inch monitor
plt.figure(figsize=(19.17,11.98))
# order controls the display order of the samples
sns.catplot(x="Sample", y="Somatic", kind="bar", data=var_counts, order=labels);
# keeps x-axis labels, but eliminates the tick mark
plt.tick_params(labelbottom=True, bottom=False)
# trim off the x-axis
sns.despine(offset=10, trim=True, bottom=True)
# labels
plt.title('')
plt.ylabel('', fontsize=8)
plt.xlabel('', fontsize=8)
# manual control of xlabels
labels = ['Indiv_1-a','Indiv_2','Indiv_3','Indiv_1-b']
# control xtick order
plt.xticks(range(len(labels)), labels, rotation=45)
# control the number of x-ticks
plt.locator_params(axis='x', nbins=10)
# legend positioning
plt.legend(loc='upper right')
# log scale
plt.gca().set_yscale('log')
# this is better if neg values are needed
plt.gca().set_yscale('symlog')
# fit plot to display
plt.tight_layout()
plt.show()
# save figure with tight_layout
plt.savefig("test.svg", format="svg", bbox_inches="tight", dpi=1000)
```

Signifance information can be added by including p-values and label bars using the following code.

```python
x1, x2 = 0, 1 # columns to annotate on the plot
y2, y1 = 20, 15 # placement of the line and how for down the vertical legs go
```

```
plt.plot([x1,x1, x2, x2], [y1, y2, y2, y1], linewidth=1, color='k') # stats line
plt.text((x1+x2)*.5, y2+2, "p=0.09", ha='center', va='bottom', fontsize=8) # p-value or sig
```

## 4.4  Statistics

This is a two-sided T-test for the null hypothesis that two populations have the same means. It is important to note that it assumes the population variances are the same, so this must be changed if the assumption is incorrect.

```
# ttest_ind(a, b, axis=0, equal_var=True, nan_policy='propagate')
from scipy.stats import ttest_ind
ttest_ind(df[df['sample'] == 'one']['means'], df[df['sample'] == 'two']['means'])
```

## 4.5  Various Plot Styles

This displays each individual datapoint overlayed on a boxplot

```
ax = sns.boxplot(x='day', y='total_bill', data=tips)
ax = sns.swarmplot(x='day', y='total_bill', data=tips, color='.25')
```

# Chapter 5

# Biology

## 5.1 General

Some helpful commands for genetic sequence.

```python
from string import ascii_uppercase # python 3
from string import upper, lower # python 2
upper('tcga')
lower('TCGA')
title('tcga') # capitalize the first letter
```

## 5.2 Biopython

Reverse complement of sequence

```python
from Bio.Seq import Seq
str(Seq(i).reverse_complement())
```

## 5.3 UCSC Genome Browser

Get sequence from UCSC genome browser

```python
from subprocess import check_output, STDOUT
temp = check_output('wget -qO- http://genome.ucsc.edu/cgi-bin/das/hg19/dna?segment=%s:%s,%s' % (vcfObj.
```

## 5.4 Ref Genome

Get sequence from reference genome

```python
from subprocess import check_output, STDOUT
temp = check_output('samtools faidx %s %s:%s-%s' % (ref, vcfObj.chrom, low, high), stderr=STDOUT, shell=
```

```python
finalSeq = ''
for line in temp.decode('UTF-8').split('\n'):
for line in temp.decode('UTF-8').split('\n'): # this is only necessary in python 3 to convert binary to
    if '>' not in line:
        finalSeq += line

finalSeq = finalSeq.upper()
```

## 5.5   Personal Information

```python
# parse vcf file with parseline
if '#' not in line and 'chr' in line: # skip the info
# vcf handling
from parseline import VCFObj
# or
from util import VCFObj
vcfObj = VCFObj(vcfLine)
# available attributes: ao, dp, af, wt, var, chrom, location
```

# Chapter 6

# Data I/O

## 6.1 Reading Data Files

Opening .gz files

```
import gzip
for line in gzip.open('myFile.gz'):
    print line
```

## 6.2 Pickles

Writing data in pickle format

```
import pickle
p = open('principle.pkl', 'wb')
pickle.dump(principleData, p)
p.close()
```

Reading data in pickle format

```
import pickle
p = open('principle.pkl', 'rb')
principleData = pickle.load(p)
p.close()
```

# Chapter 7

# Pandas

## 7.1 File I/O

Read a csv file into a DataFrame.

```
pd.read_csv(filepath)
```

Write a DataFrame to a file.

```
x.to_csv(path_or_buf='outputDir', sep='\n', header=False, index=False)
```

## 7.2 Data Structure Creation

Create a DataFrame.

```
frame = pd.DataFrame(np.random.randn(4,3), columns=list('bde'), index=['Utah','Ohio','Texas','Oregon'])
```

A DataFrame can conveniently be created from a dictionary.

```
import pandas as pd
data = {'AAA' : [4,5,6,7], 'BBB' : [10,20,30,40],'CCC' : [100,50,-30,-50]}
df2 = pd.DataFrame(data=data,index=[1,2,3,4]) #Note index starts at 1.
df2
```

```
##    AAA  BBB  CCC
## 1    4   10  100
## 2    5   20   50
## 3    6   30  -30
## 4    7   40  -50
```

## 7.3 Selection

Is data within a DataFrame found within a dictionary or list? (Instead of a dictionary a series can be used and maybe another DataFrame)

```
import pandas as pd
df = pd.DataFrame({'A': [1, 2, 3], 'B': ['a', 'b', 'f']})
df.isin([1, 3, 12, 'a'])
```

```
##        A       B
## 0   True    True
## 1  False   False
## 2   True   False
```

```
df[df.isin([1, 3, 12, 'a'])]
```

```
##       A     B
## 0   1.0     a
## 1   NaN   NaN
## 2   3.0   NaN
```

Data within a DataFrame can be selected based on position within the DataFrame.

```
import pandas as pd
df2.iloc[1:3]
```

```
##    AAA  BBB  CCC
## 2    5   20   50
## 3    6   30  -30
```

Data within a DataFrame can be selected based on position within the DataFrame.

```
import pandas as pd
df2.loc[1:3]
```

```
##    AAA  BBB  CCC
## 1    4   10  100
## 2    5   20   50
## 3    6   30  -30
```

The opposite of matching data can be selected with the inverse operator.

```
df[~((df.AAA <= 6) & (df.index.isin([0,2,4])))]
```

## 7.4   Splitting

Concatenate two DataFrames together without dropping any values or renaming indices.

```
left = pd.concat([left,left])
```

Concatenate two DataFrames together without dropping values, but renaming index.

```
left = pd.concat([left,left], ignore_index=True)
```

Count the number of each unique value in a specified column.

```
left['key1'].value_counts()
left.key1.value_counts()
```

Value counts can also be calculated as percentages so that raw counts as percent makeup can be compared.

```
left['key1'].value_counts(normalize=True) * 100
```

Two DataFrames can be merged such that only the data containing matching keys is retained.

```
result = pd.merge(left, right, how='inner', on=['key1', 'key2'])
```

This DataFrame merge will retain all of the data in the right DataFrame.

```
result = pd.merge(left, right, how='right', on=['key1', 'key2'])
```

Filter by multiple columns.

```
df[(df.one == 1) & (df.two == 2)]
```

Filter by multiple columns but only return certain values.

```
# this just returns the data in column AAA
df = pd.DataFrame({'AAA' : [4,5,6,7], 'BBB' : [10,20,30,40],'CCC' : [100,50,-30,-50]})
newseries = df.loc[(df['BBB'] < 25) & (df['CCC'] >= -40), 'AAA']
```

Filtering by values and using assignment will modify the original DataFrame.

```
df.loc[(df['BBB'] > 25) | (df['CCC'] >= 75), 'AAA'] = 0.1
```

Select multiple values from a particular column, where Letter is the column header.

```
df[df.Letter.isin(['a','b'])]
```

Use itertools to find combinations of data within a column of two DataFrames.

```
itertools.product(df1['a'], df2['a'])
```

Add data to a particular cell within a DataFrame.

```
df.loc[index,column]=num
```

Make a copy of a DataFrame.

```python
df.copy(deep=True)
```

Iterate through a DataFrame.

```python
for i in df.itertuples():
    pass
```

Change order of columns.

```python
x = x.reindex(columns=['header','seq','plus','qual'])
```

Make a DataFrame from a dictionary

```python
d = {'col1': [1, 2], 'col2': [3, 4]}
x = pd.DataFrame(d)
```

Sample from a DataFrame.

```python
df.sample(frac=1)
df.sample(n=20, axis=1)
```

Append to a DataFrame.

```python
df=df.append(newdf, ignore_index=True) # without ignore_index, the original indices will be used
```

Remove duplicates

```python
x = x[~x.index.duplicated(keep='first')] # most ideal method

data = pd.DataFrame({'k1':['one','two']*3+['two'],'k2':[1,1,2,3,3,4,4]})
data.duplicated() # identify duplicate data
data['k1'].duplicated()
data['k1'].drop_duplicates()
data.drop_duplicates['k1'] # this does the same thing as the previous line
data.drop_duplicates(['k1','k2'], keep='last') # drops unique found in k1 and k2 and keeps the last ind
```

Check if string is within strings in a given column

```python
x[x['strLoc'].str.contains(region)]
```

## 7.5   Relabeling

Rename a column or group of columns can be done by passing a dictionary of the changes.

```python
df = df.rename(columns={'a':'b','c':'d'})
```

## 7.6 Sorting and Arranging

The data in a DataFrame can be sorted in numeric or lexicographic order. The following code sorts the values within the columns a and b.

```python
df.sort_values(['a','b'], ascending=False)
```

Set a column as the new index

```python
x.set_index(['uniques'])
```

## 7.7 Editing Data

Drop columns from a DataFrame.

```python
import numpy as np
df = pd.DataFrame(np.arange(12).reshape(3,4),
                  columns=['A', 'B', 'C', 'D'])
print(df)

df = df.drop(columns=['B', 'C']) # may not work in python 2
df = df.drop(['B', 'C'], axis=1) # this works in python 2
print(df)
```

Changing the datatype of a column of data can be done by just changing column type.

```python
df.Age = df.Age.astype(str)
```

Replace values.

```python
data = pd.Series([1., -999., 2., -999., -1000., 3.])
data.replace(-999, np.nan)
```

### 7.7.1 Replace values

New data can be set within a DataFrame one subset at a time in a way that will avoid the SettingWith-CopyWarning.

```python
import pandas as pd
df = pd.DataFrame({'Trait':['Seed_Shape','Seed_Shape','Flower_Color','Flower_Color'],
                   'Phenotype':['Round','Wrinkled','Purple','White']})
df.loc[df.Trait == 'Seed_Shape', 'Affected_Part'] = 'Seed'
df.loc[df.Trait == 'Flower_Color', 'Affected_Part'] = 'Flower'
print(df)


##           Trait Phenotype Affected_Part
## 0    Seed_Shape     Round          Seed
## 1    Seed_Shape  Wrinkled          Seed
## 2  Flower_Color    Purple        Flower
## 3  Flower_Color     White        Flower
```

There is a more simple alternative to the above method buit it may result in the SettingWithCopyWarning.

```
df = df.replace('pork','bacon')
```

## 7.8   Combining Data Structures

The following merges df and df2 using inner to get the intersection on the Sample column, where indexes are ignored if the merging is performed on a column as in the following example. The other possible merging strategies are: left: use only keys from left frame, similar to a SQL left outer join; preserve key order. right: use only keys from right frame, similar to a SQL right outer join; preserve key order. outer: use union of keys from both frames, similar to a SQL full outer join; sort keys lexicographically. inner: use intersection of keys from both frames, similar to a SQL inner join; preserve the order of the left keys.

```
df = pd.merge(df, df2, how='inner', on=['Sample'])
```

Appending to a Dataframe attaches a DataFrame after another one.

```
df = pd.DataFrame([[1, 2], [3, 4]], columns=list('AB'))
df2 = pd.DataFrame([[5, 6], [7, 8]], columns=list('AB'))
df.append(df2)
```

## 7.9   Summarizing

The mean of column values can be calculated where each of the columns is grouped by the data in a specified column.

```
temp[['Sample','VAF','Var_Count']].groupby('Sample').mean()
```

## 7.10   Arithmetic and Row/Column-wise Analysis

Sometimes it is helpful to analyze the value in a particular cell in a conditional manner depending on it's value and then set the result of this analysis to a corresponding cell in a new column. Here is an example where the VAF of a variant is conditionally analyzed.

```
def LOH(x):
    if x > 0.75: return 1 - x
    elif x <= 0.75 and x > 0.25: return abs(0.5 - x)
    else: return 0
all_vars['LOH'] = all_vars.VAF.transform(LOH)
max_loh = all_vars.groupby('Sample').LOH.max().reset_index().rename(columns={'LOH':'Max_LOH'})
all_vars = pd.merge(all_vars, max_loh, how='inner', on=['Sample'])
```

Broadcasting arithmetic is an efficient method of calculating across an entire DataFrame.

```
frame = pd.DataFrame(np.arange(12.).reshape((4,3)), columns=list('bde'), index=['Utah','Ohio','Texas','(
series = frame.iloc[0]
frame - series
# the subtraction function could also be used
# frame.sub(series, axis='columns')
```

Apply a function to each row or column.

```python
f = lambda x: x.max() - x.min()
frame.apply(f, axis='index')
```

Add two sets of data together, and use fill_value to avoid replacing any missing data with `NaN`.

```python
x = pd.DataFrame([1,2,3], columns=list('0'))
y = pd.DataFrame([1,2,3], columns=list('1'))
x = x.add(y, fill_value=0)
```

Take the mean or std across specified columns and append as a new column. Below the DataFrame has columns 1-7 that will be used in computing the mean or std and this new data will be appended in a new column labeled 'Mean' or 'Std'.

```python
x['Mean']=x[[1,2,3,4,5,6,7]].mean(axis=1)
x['Std']=x[[1,2,3,4,5,6,7]].std(axis=1)
```

# Chapter 8

# Git

## 8.1 Setup

### 8.1.1 Git Setup

The username and email needs to be added after git is installed.

```
git config --global user.name "me"
git config --global user.email "me@gmail.com"
```

After this information has been set, it can be checked.

```
git config --list
```

### 8.1.2 Repository Initiation

To setup a repository, create a folder with an initial file like a README and then initiate it.

```
git init
git status
```

### 8.1.3 Mirror on Online Repository

Create a repository on a repository like github, gitlab, bitbucket, or sourceforge. Then the local git repository can be synched with the online repository.

```
git remote add origin url-of-online-repository-here
git push -u origin master
```

Of course the repository could just be setup first and then cloned.

```
git clone url-of-online-repository-here
```

## 8.2   Manipulating Commits

### 8.2.1   Repository Status

The commit history of a repository can be displayed in verbose form and in summarized form.

```
git log
git log --oneline
```

### 8.2.2   File Checkout

To restore a previous version of a file it can be checked out by first identifying the version to be used using the log history and then restoring the desired file.

```
git log --oneline
git checkout <commit number> file.txt
```

### 8.2.3   Resetting a Repository

To discard the effect of the previous operation on a file.

```
git reset HEAD file.txt
```

The previous version of the a file can then be restored.

```
git checkout -- file.txt
```

# Chapter 9

# VIM

## 9.1 Formatting

Automatic newlines are inserted by default; this behavior can be overidden with the following.

```
:set wrap
:set textwidth=0 wrapmargin=0
```

## 9.2 Spellcheck

To setup spellchecking first setup a personal dictionary file.

```
# make a directory for personal dictionary
mkdir -p ~/.vim/spell/
```

Then refer to the dictionary file within VIM, and enable spellchecking.

```
# set personal dictionary
:set spellfile=~/.vim/spell/en.utf-8.add
# turn spellcheck on
:set spell
```

Get spellcheck commands.

```
:help spell
```

Add a word to personal dictionary.

```
zg
```

Move to next and previous misspelled word.

```
]s
[s
```

Get suggestions for misspelled word.

```
z=
```

# Chapter 10

# Web APIs

## 10.1 Ensembl

The Ensembl Rest API has a number of different genomics tools.

Here is an example where the Rest API is used to get the genomic locus and amino acid change using only the protein name and amino acid position and identities.

```
content-type=application/json
wget -q --header='Content-type:application/json' 'https://rest.ensembl.org/map/translation/ENSMUSP000000
```

Alternatively python can be used to make the same call.

```
import requests, sys

server = "https://rest.ensembl.org"
ext = "/map/translation/ENSMUSP00000020991/878...879?"

r = requests.get(server+ext, headers={ "Content-Type" : "application/json"})

if not r.ok:
  r.raise_for_status()
  sys.exit()

decoded = r.json()
print(repr(decoded))
```

Here are examples getting variant effect consequences of a particular mutation.

```
wget -q --header='Content-type:application/json' 'https://rest.ensembl.org/vep/mus_musculus/hgvs/ENSMUS
```

## 10.2 UCSC Genome Browser

Here is an example where human genomic regions are converted to the orthologous mouse regions. First the human sequence is obtained.

```
wget -O - http://genome.ucsc.edu/cgi-bin/das/hg38/dna?segment=%s:%s,%s >> locs\n' % (chrom, low, high)
```

Then the above sequence is used in UCSC BLAT to find the orthologous region within mouse.

# Chapter 11

# Golang

## 11.1   Installation

Installation of linuxbrea

```
sh -c "$(curl -fsSL https://raw.githubusercontent.com/Linuxbrew/install/master/install.sh)"
# Add to path
test -d ~/.linuxbrew && eval $(~/.linuxbrew/bin/brew shellenv)
test -d /home/linuxbrew/.linuxbrew && eval $(/home/linuxbrew/.linuxbrew/bin/brew shellenv)
test -r ~/.bash_profile && echo "eval \$($(brew --prefix)/bin/brew shellenv)" >>~/.bash_profile
echo "eval \$($(brew --prefix)/bin/brew shellenv)" >>~/.profile
# debian/ubuntu dependencies
sudo apt-get install build-essential curl file git
```

LinuxBrew golang installation

```
brew install go
```

The GO PATH should then be checked; it should typically exist at ~/go but it can be checked like this

```
echo $GOPATH
```

## 11.2   Updating

```
brew install dep
brew upgrade dep
brew cask install spotify
```

## 11.3   Sample Program

Create a file called hello.go

```
package main

import "fmt"

func main() {
    fmt.Printf("hello, world\n")
}

The program can just be run with
go run main.go

Then compile the program
    go build

Importing multiple things can be done on one line separated by a semicolon
    import ("fmt"; "math")
```

## 11.4   Type conversion

Convert int to float64

```
var i int = 5
j := float64(i)
```

int to string

```
s := strconv.Itoa(97) // s == "97"
```

int64 to string

```
var n int64 = 97
s := strconv.FormatInt(n, 10) // s == "97" (decimal) in base 10
```

string to int

```
s := "97"
if n, err := strconv.Atoi(s); err == nil {
    fmt.Println(n+1)
} else {
    fmt.Println(s, "is not an integer.")
}
```

string to int64

```
s := "97"
n, err := strconv.ParseInt(s, 10, 64)
if err == nil {
    fmt.Printf("%d of type %T", n, n)
}
```

int to int64