datacamp

Python Basics

Getting started with Python Cheat Sheet

Learn Python online at www.DataCamp.com

> How to use this cheat sheet

Python is the most popular programming language in data science. It is easy to learn and comes with a wide array of powerful libraries for data analysis. This cheat sheet provides beginners and intermediate users a guide to starting using python. Use it to jump-start your journey with python. If you want more detailed Python cheat sheets, check out the following cheat sheets below:

Importing data in python

Data wrangling in pandas

> Accessing help and getting object types

```
1 + 1 # Everything after the hash symbol is ignored by Python
help(max) # Display the documentation for the max function
type('a') # Get the type of an object — this returns str
```

> Importing packages

Python packages are a collection of useful tools developed by the open-source community. They extend the capabilities of the python language. To install a new package (for example, pandas), you can go to your command prompt and type in pip install pandas. Once a package is installed, you can import it as follows.

```
import pandas # Import a package without an alias
import pandas as pd # Import a package with an alias
from pandas import DataFrame # Import an object from a package
```

> The working directory

The working directory is the default file path that python reads or saves files into. An example of the working directory is "C://file/path". The os library is needed to set and get the working directory.

```
import os # Import the operating system package
os.getcwd() # Get the current directory
os.setcwd("new/working/directory") # Set the working directory to a new file path
```

> Operators

Arithmetic operators

```
102 + 37 # Add two numbers with +          22 // 7 # Integer divide a number with //
102 - 37 # Subtract a number with -        3 ^ 4 # Raise to the power with ^
4 * 6 # Multiply two numbers with *        22 % 7 # Returns 1 # Get the remainder after
22 / 7 # Divide a number by another with /    division with %
```

Assignment operators

```
a = 5 # Assign a value to a
x[0] = 1 # Change the value of an item in a list
```

Numeric comparison operators

```
3 == 3 # Test for equality with ==         3 >= 3 # Test greater than or equal to with >=
3 != 3 # Test for inequality with !=       3 < 4 # Test less than with <
3 > 1 # Test greater than with >           3 <= 4 # Test less than or equal to with <=
```

Logical operators

```
~(2 == 2)          # Logical NOT with ~      (1 >= 1) | (1 < 1) # Logical OR with |
(1 != 1) & (1 < 1) # Logical AND with &      (1 != 1) ^ (1 < 1) # Logical XOR with ^
```

> Getting started with lists

A list is an ordered and changeable sequence of elements. It can hold integers, characters, floats, strings, and even objects.

Creating lists

```
# Create lists with [], elements separated by commas
x = [1, 3, 2]
```

List functions and methods

```
x.sorted(x) # Return a sorted copy of the list e.g., [1,2,3]
x.sort() # Sorts the list in-place (replaces x)
reversed(x) # Reverse the order of elements in x e.g., [2,3,1]
x.reversed() # Reverse the list in-place
x.count(2) # Count the number of element 2 in the list
```

Selecting list elements

Python lists are zero-indexed (the first element has index 0). For ranges, the first element is included but the last is not.

```
# Define the list
x = ['a', 'b', 'c', 'd', 'e']               x[1:3] # Select 1st (inclusive) to 3rd (exclusive)
x[0] # Select the 0th element in the list    x[2:] # Select the 2nd to the end
x[-1] # Select the last element in the list  x[:3] # Select 0th to 3rd (exclusive)
```

Concatenating lists

```
# Define the x and y lists
x = [1, 3, 6]               x + y # Returns [1, 3, 6, 10, 15, 21]
y = [10, 15, 21]            3 * x # Returns [1, 3, 6, 1, 3, 6]
```

> Getting started with dictionaries

A dictionary stores data values in key-value pairs. That is, unlike lists which are indexed by position, dictionaries are indexed by their keys, the names of which must be unique.

Creating dictionaries

```
# Create a dictionary with {}
{'a': 1, 'b': 4, 'c': 9}
```

Dictionary functions and methods

```
x = {'a': 1, 'b': 2, 'c': 3} # Define the x dictionary
x.keys() # Get the keys of a dictionary, returns dict_keys(['a', 'b', 'c'])
x.values() # Get the values of a dictionary, returns dict_values([1, 2, 3])
```

Selecting dictionary elements

```
x['a'] # 1 # Get a value from a dictionary by specifying the key
```

> NumPy arrays

NumPy is a python package for scientific computing. It provides multidimensional array objects and efficient operations on them. To import NumPy, you can run this Python code import numpy as np

Creating arrays

```
# Convert a python list to a NumPy array
np.array([1, 2, 3]) # Returns array([1, 2, 3])
# Return a sequence from start (inclusive) to end (exclusive)
np.arange(1,5) # Returns array([1, 2, 3, 4])
# Return a stepped sequence from start (inclusive) to end (exclusive)
np.arange(1,5,2) # Returns array([1, 3])
# Repeat values n times
np.repeat([1, 3, 6], 3) # Returns array([1, 1, 1, 3, 3, 3, 6, 6, 6])
# Repeat values n times
np.tile([1, 3, 6], 3) # Returns array([1, 3, 6, 1, 3, 6, 1, 3, 6])
```

> Math functions and methods

All functions take an array as the input.

```
np.log(x) # Calculate logarithm          np.quantile(x, q) # Calculate q-th quantile
np.exp(x) # Calculate exponential         np.round(x, n) # Round to n decimal places
np.max(x) # Get maximum value             np.var(x) # Calculate variance
np.min(x) # Get minimum value             np.std(x) # Calculate standard deviation
np.sum(x) # Calculate sum
np.mean(x) # Calculate mean
```

> Getting started with characters and strings

```
# Create a string with double or single quotes
"DataCamp"

# Embed a quote in string with the escape character \
"He said, \"DataCamp\""

# Create multi-line strings with triple quotes
"""
A Frame of Data
Tidy, Mine, Analyze It
Now You Have Meaning
Citation: https://mdsr-book.github.io/haikus.html
"""

str[0] # Get the character at a specific position
str[0:2] # Get a substring from starting to ending index (exclusive)
```

Combining and splitting strings

```
"Data" + "Framed" # Concatenate strings with +, this returns 'DataFramed'
3 * "data" # Repeat strings with *, this returns 'data data data'
"beekeepers".split("e") # Split a string on a delimiter, returns ['b', '', 'k', '', 'p', 'rs']
```

Mutate strings

```
str = "Jack and Jill" # Define str
str.upper() # Convert a string to uppercase, returns 'JACK AND JILL'
str.lower() # Convert a string to lowercase, returns 'jack and jill'
str.title() # Convert a string to title case, returns 'Jack And Jill'
str.replace("J", "P") # Replaces matches of a substring with another, returns 'Pack and Pill'
```

> Getting started with DataFrames

Pandas is a fast and powerful package for data analysis and manipulation in python. To import the package, you can use import pandas as pd. A pandas DataFrame is a structure that contains two-dimensional data stored as rows and columns. A pandas series is a structure that contains one-dimensional data.

Creating DataFrames

```
# Create a dataframe from a dictionary        # Create a dataframe from a list of dictionaries
pd.DataFrame({                                pd.DataFrame([
  'a': [1, 2, 3],                               {'a': 1, 'b': 4, 'c': 'x'},
  'b': np.array([4, 4, 6]),                     {'a': 2, 'b': 4, 'c': 'x'},
  'c': ['x', 'x', 'y']                          {'a': 3, 'b': 6, 'c': 'y'}
})                                            ])
```

Selecting DataFrame Elements

Select a row, column or element from a dataframe. Remember: all positions are counted from zero, not one.

```
# Select the 3rd row
df.iloc[3]
# Select one column by name
df['col']
# Select multiple columns by names
df[['col1', 'col2']]
# Select 2nd column
df.iloc[:, 2]
# Select the element in the 3rd row, 2nd column
df.iloc[3, 2]
```

Manipulating DataFrames

```
# Concatenate DataFrames vertically           # Calculate the mean of each column
pd.concat([df, df])                            df.mean()
# Concatenate DataFrames horizontally          # Get summary statistics by column
pd.concat([df,df],axis="columns")              df.agg(aggregation_function)
# Get rows matching a condition                # Get unique rows
df.query('logical_condition')                  df.drop_duplicates()
# Drop columns by name                         # Sort by values in a column
df.drop(columns=['col_name'])                  df.sort_values(by='col_name')
# Rename columns                               # Get rows with largest values in a column
df.rename(columns={"oldname": "newname"})      df.nlargest(n, 'col_name')
# Add a new column
df.assign(temp_f=9 / 5 * df['temp_c'] + 32)
```

Python For Data Science
Importing Data Cheat Sheet

Learn Python online at www.DataCamp.com

> Importing Data in Python

Most of the time, you'll use either NumPy or pandas to import your data:

```
>>> import numpy as np
>>> import pandas as pd
```

> Help

```
>>> np.info(np.ndarray.dtype)
>>> help(pd.read_csv)
```

> Text Files

Plain Text Files

```
>>> filename = 'huck_finn.txt'
>>> file = open(filename, mode='r') #Open the file for reading
>>> text = file.read() #Read a file's contents
>>> print(file.closed) #Check whether file is closed
>>> file.close() #Close file
>>> print(text)
```

Using the context manager with

```
>>> with open('huck_finn.txt', 'r') as file:
        print(file.readline()) #Read a single line
        print(file.readline())
        print(file.readline())
```

Table Data: Flat Files

Importing Flat Files with NumPy

```
>>> filename = 'huck_finn.txt'
>>> file = open(filename, mode='r') #Open the file for reading
>>> text = file.read() #Read a file's contents
>>> print(file.closed) #Check whether file is closed
>>> file.close() #Close file
>>> print(text)
```

Files with one data type

```
>>> filename = 'mnist.txt'
>>> data = np.loadtxt(filename,
                      delimiter=',', #String used to separate values
                      skiprows=2, #Skip the first 2 lines
                      usecols=[0,2], #Read the 1st and 3rd column
                      dtype=str) #The type of the resulting array
```

Files with mixed data type

```
>>> filename = 'titanic.csv'
>>> data = np.genfromtxt(filename,
                         delimiter=',',
                         names=True, #Look for column header
                         dtype=None)
>>> data_array = np.recfromcsv(filename)
#The default dtype of the np.recfromcsv() function is None
```

Importing Flat Files with Pandas

```
>>> filename = 'winequality-red.csv'
>>> data = pd.read_csv(filename,
                       nrows=5, #Number of rows of file to read
                       header=None, #Row number to use as col names
                       sep='\t', #Delimiter to use
                       comment='#', #Character to split comments
                       na_values=[""]) #String to recognize as NA/NaN
```

> Exploring Your Data

NumPy Arrays

```
>>> data_array.dtype #Data type of array elements
>>> data_array.shape #Array dimensions
>>> len(data_array) #Length of array
```

Pandas DataFrames

```
>>> df.head() #Return first DataFrame rows
>>> df.tail() #Return last DataFrame rows
>>> df.index #Describe index
>>> df.columns #Describe DataFrame columns
>>> df.info() #Info on DataFrame
>>> data_array = data.values #Convert a DataFrame to an a NumPy array
```

> SAS File

```
>>> from sas7bdat import SAS7BDAT
>>> with SAS7BDAT('urbanpop.sas7bdat') as file:
        df_sas = file.to_data_frame()
```

> Stata File

```
>>> data = pd.read_stata('urbanpop.dta')
```

> Excel Spreadsheets

```
>>> file = 'urbanpop.xlsx'
>>> data = pd.ExcelFile(file)
>>> df_sheet2 = data.parse('1960-1966',
                           skiprows=[0],
                           names=['Country',
                           'AAM: War(2002)'])
>>> df_sheet1 = data.parse(0,
                           parse_cols=[0],
                           skiprows=[0],
                           names=['Country'])
```

To access the sheet names, use the sheet_names attribute:

```
>>> data.sheet_names
```

> Relational Databases

```
>>> from sqlalchemy import create_engine
>>> engine = create_engine('sqlite:///Northwind.sqlite')
```

Use the table_names() method to fetch a list of table names:

```
>>> table_names = engine.table_names()
```

Querying Relational Databases

```
>>> con = engine.connect()
>>> rs = con.execute("SELECT * FROM Orders")
>>> df = pd.DataFrame(rs.fetchall())
>>> df.columns = rs.keys()
>>> con.close()
```

Using the context manager with

```
>>> with engine.connect() as con:
        rs = con.execute("SELECT OrderID FROM Orders")
        df = pd.DataFrame(rs.fetchmany(size=5))
        df.columns = rs.keys()
```

Querying relational databases with pandas

```
>>> df = pd.read_sql_query("SELECT * FROM Orders", engine)
```

> Pickled Files

```
>>> import pickle
>>> with open('pickled_fruit.pkl', 'rb') as file:
        pickled_data = pickle.load(file)
```

> Matlab Files

```
>>> import scipy.io
>>> filename = 'workspace.mat'
>>> mat = scipy.io.loadmat(filename)
```

> HDF5 Files

```
>>> import h5py
>>> filename = 'H-H1_LOSC_4_v1-815411200-4096.hdf5'
>>> data = h5py.File(filename, 'r')
```

> Exploring Dictionaries

Querying relational databases with pandas

```
>>> print(mat.keys()) #Print dictionary keys
>>> for key in data.keys(): #Print dictionary keys
        print(key)
meta
quality
strain
>>> pickled_data.values() #Return dictionary values
>>> print(mat.items()) #Returns items in list format of (key, value) tuple pairs
```

Accessing Data Items with Keys

```
>>> for key in data ['meta'].keys() #Explore the HDF5
structure
        print(key)
Description
DescriptionURL
Detector
Duration
GPSstart
Observatory
Type
UTCstart
#Retrieve the value for a key
>>> print(data['meta']['Description'].value)
```
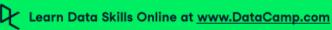
> Navigating Your FileSystem

Magic Commands

```
!ls  #List directory contents of files and directories
%cd .. #Change current working directory
%pwd #Return the current working directory path
```

OS Library

```
>>> import os
>>> path = "/usr/tmp"
>>> wd = os.getcwd() #Store the name of current directory in a string
>>> os.listdir(wd) #Output contents of the directory in a list
>>> os.chdir(path) #Change current working directory
>>> os.rename("test1.txt", #Rename a file
              "test2.txt")
>>> os.remove("test1.txt") #Delete an existing file
>>> os.mkdir("newdir") #Create a new directory
```

datacamp

Python For Data Science
PySpark RDD Cheat Sheet

Learn PySpark RDD online at www.DataCamp.com

Spark

PySpark is the Spark Python API that exposes the Spark programming model to Python.

> Initializing Spark

SparkContext

```
>>> from pyspark import SparkContext
>>> sc = SparkContext(master = 'local[2]')
```

Inspect SparkContext

```
>>> sc.version              #Retrieve SparkContext version
>>> sc.pythonVer            #Retrieve Python version
>>> sc.master               #Master URL to connect to
>>> str(sc.sparkHome)       #Path where Spark is installed on worker nodes
>>> str(sc.sparkUser())     #Retrieve name of the Spark User running SparkContext
>>> sc.appName              #Return application name
>>> sc.applicationId        #Retrieve application ID
>>> sc.defaultParallelism   #Return default level of parallelism
>>> sc.defaultMinPartitions #Default minimum number of partitions for RDDs
```

Configuration

```
>>> from pyspark import SparkConf, SparkContext
>>> conf = (SparkConf()
        .setMaster("local")
        .setAppName("My app")
        .set("spark.executor.memory", "1g"))
>>> sc = SparkContext(conf = conf)
```

Using The Shell

In the PySpark shell, a special interpreter-aware SparkContext is already created in the variable called `sc`.

```
$ ./bin/spark-shell --master local[2]
$ ./bin/pyspark --master local[4] --py-files code.py
```

Set which master the context connects to with the `--master` argument, and add Python .zip, .egg or .py files to the runtime path by passing a comma-separated list to `--py-files`.

> Loading Data

Parallelized Collections

```
>>> rdd = sc.parallelize([('a',7),('a',2),('b',2)])
>>> rdd2 = sc.parallelize([('a',2),('d',1),('b',1)])
>>> rdd3 = sc.parallelize(range(100))
>>> rdd4 = sc.parallelize([("a",["x","y","z"]),
                            ("b",["p", "r"])])
```

External Data

Read either one text file from HDFS, a local file system or any Hadoop-supported file system URI with `textFile()`, or read in a directory of text files with `wholeTextFiles()`.

```
>>> textFile = sc.textFile("/my/directory/*.txt")
>>> textFile2 = sc.wholeTextFiles("/my/directory/")
```

> Retrieving RDD Information

Basic Information

```
>>> rdd.getNumPartitions()  #List the number of partitions
>>> rdd.count()             #Count RDD instances 3
>>> rdd.countByKey()        #Count RDD instances by key
 defaultdict(<type 'int'>,{'a':2,'b':1})
>>> rdd.countByValue()      #Count RDD instances by value
 defaultdict(<type 'int'>,{('b',2):1,('a',2):1,('a',7):1})
>>> rdd.collectAsMap()      #Return (key,value) pairs as a dictionary
 {'a': 2,'b': 2}
>>> rdd3.sum()              #Sum of RDD elements 4950
>>> sc.parallelize([]).isEmpty()  #Check whether RDD is empty
 True
```

Summary

```
>>> rdd3.max()      #Maximum value of RDD elements
 99
>>> rdd3.min()      #Minimum value of RDD elements
 0
>>> rdd3.mean()     #Mean value of RDD elements
 49.5
>>> rdd3.stdev()    #Standard deviation of RDD elements
 28.866070047722118
>>> rdd3.variance() #Compute variance of RDD elements
 833.25
>>> rdd3.histogram(3) #Compute histogram by bins
 ([0,33,66,99],[33,33,34])
>>> rdd3.stats()    #Summary statistics (count, mean, stdev, max & min)
```

> Applying Functions

```
#Apply a function to each RDD element
>>> rdd.map(lambda x: x+(x[1],x[0])).collect()
 [('a',7,7,'a'),('a',2,2,'a'),('b',2,2,'b')]
#Apply a function to each RDD element and flatten the result
>>> rdd5 = rdd.flatMap(lambda x: x+(x[1],x[0]))
>>> rdd5.collect()
 ['a',7,7,'a','a',2,2,'a','b',2,2,'b']
#Apply a flatMap function to each (key,value) pair of rdd4 without changing the keys
>>> rdd4.flatMapValues(lambda x: x).collect()
 [('a','x'),('a','y'),('a','z'),('b','p'),('b','r')]
```

> Selecting Data

Getting

```
>>> rdd.collect()  #Return a list with all RDD elements
 [('a', 7), ('a', 2), ('b', 2)]
>>> rdd.take(2)    #Take first 2 RDD elements
 [('a', 7), ('a', 2)]
>>> rdd.first()    #Take first RDD element
 ('a', 7)
>>> rdd.top(2)     #Take top 2 RDD elements
 [('b', 2), ('a', 7)]
```

Sampling

```
>>> rdd3.sample(False, 0.15, 81).collect()  #Return sampled subset of rdd3
 [3,4,27,31,40,41,42,43,60,76,79,80,86,97]
```

Filtering

```
>>> rdd.filter(lambda x: "a" in x).collect()  #Filter the RDD
 [('a',7),('a',2)]
>>> rdd5.distinct().collect()  #Return distinct RDD values
 ['a',2,'b',7]
>>> rdd.keys().collect()  #Return (key,value) RDD's keys
 ['a', 'a', 'b']
```

> Iterating

```
>>> def g(x): print(x)
>>> rdd.foreach(g)  #Apply a function to all RDD elements
 ('a', 7)
 ('b', 2)
 ('a', 2)
```

> Reshaping Data

Reducing

```
>>> rdd.reduceByKey(lambda x,y : x+y).collect()  #Merge the rdd values for each key
 [('a',9),('b',2)]
>>> rdd.reduce(lambda a, b: a + b)  #Merge the rdd values
 ('a',7,'a',2,'b',2)
```

Grouping by

```
>>> rdd3.groupBy(lambda x: x % 2)  #Return RDD of grouped values
        .mapValues(list)
        .collect()
>>> rdd.groupByKey()  #Group rdd by key
        .mapValues(list)
        .collect()
 [('a',[7,2]),('b',[2])]
```

Aggregating

```
>>> seqOp = (lambda x,y: (x[0]+y,x[1]+1))
>>> combOp = (lambda x,y:(x[0]+y[0],x[1]+y[1]))
#Aggregate RDD elements of each partition and then the results
>>> rdd3.aggregate((0,0),seqOp,combOp)
 (4950,100)
#Aggregate values of each RDD key
>>> rdd.aggregateByKey((0,0),seqop,combop).collect()
 [('a',(9,2)), ('b',(2,1))]
#Aggregate the elements of each partition, and then the results
>>> rdd3.fold(0,add)
 4950
#Merge the values for each key
>>> rdd.foldByKey(0, add).collect()
 [('a',9),('b',2)]
#Create tuples of RDD elements by applying a function
>>> rdd3.keyBy(lambda x: x+x).collect()
```

> Mathematical Operations

```
>>> rdd.subtract(rdd2).collect()  #Return each rdd value not contained in rdd2
 [('b',2),('a',7)]
#Return each (key,value) pair of rdd2  with no matching key in rdd
>>> rdd2.subtractByKey(rdd).collect()
 [('d', 1)]
>>> rdd.cartesian(rdd2).collect()  #Return the Cartesian product of rdd and rdd2
```

> Sort

```
>>> rdd2.sortBy(lambda x: x[1]).collect()  #Sort RDD by given function
 [('d',1),('b',1),('a',2)]
>>> rdd2.sortByKey().collect()  #Sort (key, value) RDD by key
 [('a',2),('b',1),('d',1)]
```

> Repartitioning

```
>>> rdd.repartition(4)  #New RDD with 4 partitions
>>> rdd.coalesce(1)  #Decrease the number of partitions in the RDD to 1
```

> Saving

```
>>> rdd.saveAsTextFile("rdd.txt")
>>> rdd.saveAsHadoopFile("hdfs://namenodehost/parent/child",
                         'org.apache.hadoop.mapred.TextOutputFormat')
```

> Stopping SparkContext

```
>>> sc.stop()
```

> Execution

```
$ ./bin/spark-submit examples/src/main/python/pi.py
```

Learn Data Skills Online at www.DataCamp.com

Working with text data in Python

Learn Python online at www.DataCamp.com

> Example data used throughout this cheat sheet

Throughout this cheat sheet, we'll be using two pandas series named suits and rock_paper_scissors.

```python
import pandas as pd

suits = pd.Series(["clubs", "Diamonds", "hearts", "Spades"])
rock_paper_scissors = pd.Series(["rock ", " paper", "scissors"])
```

> String lengths and substrings

```python
# Get the number of characters with .str.len()
suits.str.len() # Returns 5 8 6 6

# Get substrings by position with .str[]
suits.str[2:5] # Returns "ubs" "amo" "art" "ade"

# Get substrings by negative position with .str[]
suits.str[:-3] # "cl" "Diamo" "hea" "Spa"

# Remove whitespace from the start/end with .str.strip()
rock_paper_scissors.str.strip() # "rock" "paper" "scissors"

# Pad strings to a given length with .str.pad()
suits.str.pad(8, fillchar="_") # "___clubs" "Diamonds" "__hearts" "__Spades"
```

> Changing case

```python
# Convert to lowercase with .str.lower()
suits.str.lower() # "clubs" "diamonds" "hearts" "spades"

# Convert to uppercase with .str.upper()
suits.str.upper() # "CLUBS" "DIAMONDS" "HEARTS" "SPADES"

# Convert to title case with .str.title()
pd.Series("hello, world!").str.title() # "Hello, World!"

# Convert to sentence case with .str.capitalize()
pd.Series("hello, world!").str.capitalize() # "Hello, world!"
```

> Formatting settings

```python
# Generate an example DataFramed named df
df = pd.DataFrame({"x": [0.123, 4.567, 8.901]})
#     x
# 0 0.123
# 1 4.567
# 2 8.901
```

```python
# Visualize and format table output
df.style.format(precision = 1)
```

-	x
0	0.1
1	4.5
2	8.9

The output of style.format is an HTML table

> Splitting strings

```python
# Split strings into list of characters with .str.split(pat="")
suits.str.split(pat="")

# [, "c" "l" "u" "b" "s", ]
# [, "D" "i" "a" "m" "o" "n" "d" "s", ]
# [, "h" "e" "a" "r" "t" "s", ]
# [, "S" "p" "a" "d" "e" "s", ]

# Split strings by a separator with .str.split()
suits.str.split(pat = "a")

# ["clubs"]
# ["Di", "monds"]
# ["he", "rts"]
# ["Sp", "des"]

# Split strings and return DataFrame with .str.split(expand=True)
suits.str.split(pat = "a", expand=True)

#       0      1
# 0  clubs   None
# 1     Di  monds
# 2     he    rts
# 3     Sp    des
```

> Joining or concatenating strings

```python
# Combine two strings with +
suits + "5" # "clubs5" "Diamonds5" "hearts5" "Spades5"

# Collapse character vector to string with .str.cat()
suits.str.cat(sep=", ) # "clubs, Diamonds, hearts, Spades"

# Duplicate and concatenate strings with *
suits * 2 # "clubsclubs" "DiamondsDiamonds" "heartshearts" "SpadesSpades"
```

> Detecting Matches

```python
# Detect if a regex pattern is present in strings with .str.contains()
suits.str.contains("[ae]") # False True True True

# Count the number of matches with .str.count()
suits.str.count("[ae]") # 0 1 2 2

# Locate the position of substrings with str.find()
suits.str.find("e") # -1 -1 1 4
```

> Extracting matches

```python
# Extract matches from strings with str.findall()
suits.str.findall(".[ae]") # [] ["ia"] ["he"[ ["pa", "de"]

# Extract capture groups with .str.extractall()
suits.str.extractall("([ae])(.)")
#         0 1
#     match
# 1 0     a m
# 2 0     e a
# 3 0     a d
#   1     e s

# Get subset of strings that match with x[x.str.contains()]
suits[suits.str.contains("d")] # "Diamonds" "Spades"
```

> Replacing matches

```python
# Replace a regex match with another string with .str.replace()
suits.str.replace("a", "4") # "clubs" "Di4monds" "he4rts" "Sp4des"

# Remove a suffix with .str.removesuffix()
suits.str.removesuffix # "club" "Diamond" "heart" "Spade"

# Replace a substring with .str.slice_replace()
rhymes = pd.Series(["vein", "gain", "deign"])
rhymes.str.slice_replace(0, 1, "r") # "rein" "rain" "reign"
```

datacamp

Python For Data Science
python Basics Cheat Sheet

Learn Python Basics online at www.DataCamp.com

> Variables and Data Types

Variable Assignment

```
>>> x=5
>>> x
5
```

Calculations With Variables

```
>>> x+2 #Sum of two variables
7
>>> x-2 #Subtraction of two variables
3
>>> x*2 #Multiplication of two variables
10
>>> x**2 #Exponentiation of a variable
25
>>> x%2 #Remainder of a variable
1
>>> x/float(2) #Division of a variable
2.5
```

Types and Type Conversion

```
str()
'5', '3.45', 'True' #Variables to strings

int()
5, 3, 1 #Variables to integers

float()
5.0, 1.0 #Variables to floats

bool()
True, True, True #Variables to booleans
```

> Libraries

pandas — Data analysis
NumPy — Scientific computing
matplotlib — 2D plotting
Machine learning

Import Libraries

```
>>> import numpy
>>> import numpy as np
```

Selective import

```
>>> from math import pi
```

> Strings

```
>>> my_string = 'thisStringIsAwesome'
>>> my_string
'thisStringIsAwesome'
```

String Operations

```
>>> my_string * 2
'thisStringIsAwesomethisStringIsAwesome'
>>> my_string + 'Innit'
'thisStringIsAwesomeInnit'
>>> 'm' in my_string
True
```

String Indexing   Index starts at 0

```
>>> my_string[3]
>>> my_string[4:9]
```

String Methods

```
>>> my_string.upper() #String to uppercase
>>> my_string.lower() #String to lowercase
>>> my_string.count('w') #Count String elements
>>> my_string.replace('e', 'i') #Replace String elements
>>> my_string.strip() #Strip whitespaces
```

> NumPy Arrays   Also see Lists

```
>>> my_list = [1, 2, 3, 4]
>>> my_array = np.array(my_list)
>>> my_2darray = np.array([[1,2,3],[4,5,6]])
```

Selecting Numpy Array Elements   Index starts at 0

```
Subset
>>> my_array[1] #Select item at index 1
2

Slice
>>> my_array[0:2] #Select items at index 0 and 1
array([1, 2])

Subset 2D Numpy arrays
>>> my_2darray[:,0] #my_2darray[rows, columns]
array([1, 4])
```

Numpy Array Operations

```
>>> my_array > 3
array([False, False, False, True], dtype=bool)
>>> my_array * 2
array([2, 4, 6, 8])
>>> my_array + np.array([5, 6, 7, 8])
array([6, 8, 10, 12])
```

Numpy Array Functions

```
>>> my_array.shape #Get the dimensions of the array
>>> np.append(other_array) #Append items to an array
>>> np.insert(my_array, 1, 5) #Insert items in an array
>>> np.delete(my_array,[1]) #Delete items in an array
>>> np.mean(my_array) #Mean of the array
>>> np.median(my_array) #Median of the array
>>> my_array.corrcoef() #Correlation coefficient
>>> np.std(my_array) #Standard deviation
```

> Lists   Also see NumPy Arrays

```
>>> a = 'is'
>>> b = 'nice'
>>> my_list = ['my', 'list', a, b]
>>> my_list2 = [[4,5,6,7], [3,4,5,6]]
```

Selecting List Elements   Index starts at 0

```
Subset
>>> my_list[1] #Select item at index 1
>>> my_list[-3] #Select 3rd last item

Slice
>>> my_list[1:3] #Select items at index 1 and 2
>>> my_list[1:] #Select items after index 0
>>> my_list[:3] #Select items before index 3
>>> my_list[:] #Copy my_list

Subset Lists of Lists
>>> my_list2[1][0] #my_list[list][itemOfList]
>>> my_list2[1][:2]
```

List Operations

```
>>> my_list + my_list
['my', 'list', 'is', 'nice', 'my', 'list', 'is', 'nice']
>>> my_list * 2
['my', 'list', 'is', 'nice', 'my', 'list', 'is', 'nice']
>>> my_list2 > 4
True
```

List Methods

```
>>> my_list.index(a) #Get the index of an item
>>> my_list.count(a) #Count an item
>>> my_list.append('!') #Append an item at a time
>>> my_list.remove('!') #Remove an item
>>> del(my_list[0:1]) #Remove an item
>>> my_list.reverse() #Reverse the list
>>> my_list.extend('!') #Append an item
>>> my_list.pop(-1) #Remove an item
>>> my_list.insert(0,'!') #Insert an item
>>> my_list.sort() #Sort the list
```

> Python IDEs (Integrated Development Environment)

ANACONDA — Leading open data science platform powered by Python

SPYDER — Free IDE that is included with Anaconda

Jupyter — Create and share documents with live code

> Asking For Help

```
>>> help(str)
```

---

datacamp

Python For Data Science
Seaborn Cheat Sheet

Learn Seaborn online at www.DataCamp.com

Statistical Data Visualization With Seaborn

The Python visualization library **Seaborn** is based on matplotlib and provides a high-level interface for drawing attractive statistical graphics.

Make use of the following aliases to import the libraries:

```
>>> import matplotlib.pyplot as plt
>>> import seaborn as sns
```

The basic steps to creating plots with Seaborn are:
1. Prepare some data
2. Control figure aesthetics
3. Plot with Seaborn
4. Further customize your plot
5. Show your plot

```
>>> import matplotlib.pyplot as plt
>>> import seaborn as sns
>>> tips = sns.load_dataset("tips") #Step 1
>>> sns.set_style("whitegrid") #Step 2
>>> g = sns.lmplot(x="tip", #Step 3
                   y="total_bill",
                   data=tips,
                   aspect=2)
>>> g = (g.set_axis_labels("Tip","Total bill(USD)").
set(xlim=(0,10),ylim=(0,100)))
>>> plt.title("title") #Step 4
>>> plt.show(g) #Step 5
```

---

1 Data   Also see Lists, NumPy & Pandas

```
>>> import pandas as pd
>>> import numpy as np
>>> uniform_data = np.random.rand(10, 12)
>>> data = pd.DataFrame({'x':np.arange(1,101),
                         'y':np.random.normal(0,4,100)})
```

Seaborn also offers built-in data sets:

```
>>> titanic = sns.load_dataset("titanic")
>>> iris = sns.load_dataset("iris")
```

---

2 Figure Aesthetics   Also see Matplotlib

```
>>> f, ax = plt.subplots(figsize=(5,6)) #Create a figure and one subplot
```

Seaborn styles

```
>>> sns.set() #(Re)set the seaborn default
>>> sns.set_style("whitegrid") #Set the matplotlib parameters
>>> sns.set_style("ticks", #Set the matplotlib parameters
                  {"xtick.major.size":8,
                   "ytick.major.size":8})
#Return a dict of params or use with with to temporarily set the style
>>> sns.axes_style("whitegrid")
```

Context Functions

```
>>> sns.set_context("talk") #Set context to "talk"
>>> sns.set_context("notebook", #Set context to "notebook",
                    font_scale=1.5, #Scale font elements and
                    rc={"lines.linewidth":2.5}) #override param mapping
```

Color Palette

```
>>> sns.set_palette("husl",3) #Define the color palette
>>> sns.color_palette("husl") #Use with with to temporarily set palette
>>> flatui = ["#9b59b6","#3498db","#95a5a6","#e74c3c","#34495e","#2ecc71"]
>>> sns.set_palette(flatui) #Set your own color palette
```

---

3 Plotting With Seaborn

Axis Grids

```
>>> g = sns.FacetGrid(titanic, #Subplot grid for plotting conditional relationships
                      col="survived",
                      row="sex")
>>> g = g.map(plt.hist,"age")
>>> sns.factorplot(x="pclass", #Draw a categorical plot onto a Facetgrid
                   y="survived",
                   hue="sex",
                   data=titanic)
>>> sns.lmplot(x="sepal_width", #Plot data and regression model fits across a FacetGrid
               y="sepal_length",
               hue="species",
               data=iris)
>>> h = sns.PairGrid(iris) #Subplot grid for plotting pairwise relationships
>>> h = h.map(plt.scatter)
>>> sns.pairplot(iris) #Plot pairwise bivariate distributions
>>> i = sns.JointGrid(x="x", #Grid for bivariate plot with marginal univariate plots
                      y="y",
                      data=data)
>>> i = i.plot(sns.regplot,
               sns.distplot)
>>> sns.jointplot("sepal_length", #Plot bivariate distribution
                  "sepal_width",
                  data=iris,
                  kind='kde')
```

4 Further Customizations   Also see Matplotlib

Axisgrid Objects

```
>>> g.despine(left=True) #Remove left spine
>>> g.set_ylabels("Survived") #Set the labels of the y-axis
>>> g.set_xticklabels(rotation=45) #Set the tick labels for x
>>> g.set_axis_labels("Survived", #Set the axis labels
                      "Sex")
>>> h.set(xlim=(0,5), #Set the limit and ticks of the x-and y-axis
          ylim=(0,5),
          xticks=[0,2.5,5],
          yticks=[0,2.5,5])
```

Plot

```
>>> plt.title("A Title") #Add plot title
>>> plt.ylabel("Survived") #Adjust the label of the y-axis
>>> plt.xlabel("Sex") #Adjust the label of the x-axis
>>> plt.ylim(0,100) #Adjust the limits of the y-axis
>>> plt.xlim(0,10) #Adjust the limits of the x-axis
>>> plt.setp(ax,yticks=[0,5]) #Adjust a plot property
>>> plt.tight_layout() #Adjust subplot params
```

Regression Plots

```
>>> sns.regplot(x="sepal_width", #Plot data and a linear regression model fit
                y="sepal_length",
                data=iris,
                ax=ax)
```

Distribution Plots

```
>>> plot = sns.distplot(data.y, #Plot univariate distribution
                        kde=False,
                        color="b")
```

Matrix Plots

```
>>> sns.heatmap(uniform_data,vmin=0,vmax=1) #Heatmap
```

Categorical Plots

Scatterplot
```
>>> sns.stripplot(x="species", #Scatterplot with one categorical variable
                  y="petal_length",
                  data=iris)
>>> sns.swarmplot(x="species", #Categorical scatterplot with non-overlapping points
                  y="petal_length",
                  data=iris)
```

Bar Chart
```
>>> sns.barplot(x="sex", #Show point estimates & confidence intervals with scatterplot glyphs
                y="survived",
                hue="class",
                data=titanic)
```

Count Plot
```
>>> sns.countplot(x="deck", #Show count of observations
                  data=titanic,
                  palette="Greens_d")
```

Point Plot
```
>>> sns.pointplot(x="class", #Show point estimates & confidence intervals as rectangular bars
                  y="survived",
                  hue="sex",
                  data=titanic,
                  palette={"male":"g",
                  "female":"m"},
                  markers=["^","o"],
                  linestyles=["-","--"])
```

Boxplot
```
>>> sns.boxplot(x="alive", #Boxplot
                y="age",
                hue="adult_male",
                data=titanic)
>>> sns.boxplot(data=iris,orient="h") #Boxplot with wide-form data
```

Violinplot
```
>>> sns.violinplot(x="age", #Violin plot
                   y="sex",
                   hue="survived",
                   data=titanic)
```

---

5 Show or Save Plot   Also see Matplotlib

```
>>> plt.show() #Show the plot
>>> plt.savefig("foo.png") #Save the plot as a figure
>>> plt.savefig("foo.png", #Save transparent figure
                transparent=True)
```

> Close & Clear   Also see Matplotlib

```
>>> plt.cla() #Clear an axis
>>> plt.clf() #Clear an entire figure
>>> plt.close() #Close a window
```

Learn Data Skills Online at www.DataCamp.com