Python Cheatsheet

Contents

[Python Cheatsheet 1](#_Toc521589473)

[Contents 1](#_Toc521589474)

[Arrays (numpy) 11](#_Toc521589475)

[Add a column to an array 11](#_Toc521589476)

[Add a dimension to an array 11](#_Toc521589477)

[Apply a function to every element of an array 11](#_Toc521589478)

[Change the Data Type of an Array 11](#_Toc521589479)

[Concatenate Arrays 12](#_Toc521589480)

[Convert a DataFrame to an Array 12](#_Toc521589481)

[Convert a list to an array 12](#_Toc521589482)

[Count the number of equal items in two arrays 12](#_Toc521589483)

[Create an array of constants 12](#_Toc521589484)

[Create an array of zeros 13](#_Toc521589485)

[2D 13](#_Toc521589486)

[Create and add rows to an array 13](#_Toc521589487)

[Create a 2-D Array From Scratch 13](#_Toc521589488)

[Dimensions of an array 13](#_Toc521589489)

[Dot Product of 2 arrays 14](#_Toc521589490)

[Element-wise Multiplication 14](#_Toc521589491)

[Expand the Rank (number of dimensions) of an array 14](#_Toc521589492)

[Flatten a numpy Array 14](#_Toc521589493)

[Load a CSV file into a numpy array 15](#_Toc521589494)

[Normalize Rows of an ndarray 15](#_Toc521589495)

[print() options 15](#_Toc521589496)

[print entire array 15](#_Toc521589497)

[Reshape an array 15](#_Toc521589498)

[Select a Subset of Columns 15](#_Toc521589499)

[Select a Subset of Rows 16](#_Toc521589500)

[Select Rows from an Array which Match a List of Values for some Column 16](#_Toc521589501)

[Shuffle an Array 17](#_Toc521589502)

[Slicing an Array 17](#_Toc521589503)

[Return the last 3 elements 17](#_Toc521589504)

[Standardize an Array 17](#_Toc521589505)

[Statistics for Arrays 17](#_Toc521589506)

[Median for Arrays 17](#_Toc521589507)

[Sort a numpy.ndarray 18](#_Toc521589508)

[Split an Array into 2 parts 18](#_Toc521589509)

[Transpose a numpy.ndarray 18](#_Toc521589510)

[Classes 19](#_Toc521589511)

[Class static methods 19](#_Toc521589512)

[Control Statements 20](#_Toc521589513)

[for-next loops 20](#_Toc521589514)

[Count backwards 20](#_Toc521589515)

[for-next over multiple variables 20](#_Toc521589516)

[while loops 20](#_Toc521589517)

[CSV Files 21](#_Toc521589518)

[Read from a CSV file 21](#_Toc521589519)

[Write to a CSV file 21](#_Toc521589520)

[Database Functions 21](#_Toc521589521)

[Postgres 21](#_Toc521589522)

[Connect to a Postgres database 21](#_Toc521589523)

[Execute SQL query on Postgres 22](#_Toc521589524)

[SQLite3 22](#_Toc521589525)

[Create a SQLite3 Database 22](#_Toc521589526)

[Data Types 22](#_Toc521589527)

[Insert Values into Database 23](#_Toc521589528)

[Read from a Database Table 23](#_Toc521589529)

[Parameterized Queries 23](#_Toc521589530)

[DataFrame (pandas) 24](#_Toc521589531)

[Add a column to a DataFrame 24](#_Toc521589532)

[Add a row to a DataFrame 25](#_Toc521589533)

[Apply a Lambda to Every Row of a DataFrame 25](#_Toc521589534)

[Apply a function, with arguments 25](#_Toc521589535)

[Change column names 26](#_Toc521589536)

[Change Column Data Type 27](#_Toc521589537)

[Change values in one column based on values in a different column 27](#_Toc521589538)

[Concatenate two data frames 28](#_Toc521589539)

[Convert a DataFrame Column to type datetime 28](#_Toc521589540)

[Convert a Single DataFrame Column to a numpy.ndarray 28](#_Toc521589541)

[Convert a DataFrame to a numpy.ndarray 29](#_Toc521589542)

[Convert a Dict to a DataFrame 29](#_Toc521589543)

[Copy a column from another DataFrame 29](#_Toc521589544)

[Copy a DataFrame 30](#_Toc521589545)

[Correlation between columns 30](#_Toc521589546)

[Count the distinct values in a DataFrame column 30](#_Toc521589547)

[Create a DataFrame from scratch 31](#_Toc521589548)

[Create a DataFrame which has only one column 31](#_Toc521589549)

[Create a DataFrame with integer columns 31](#_Toc521589550)

[Create a Pivot Table from a DataFrame 33](#_Toc521589551)

[Delete a Column 34](#_Toc521589552)

[Delete Rows Having Nulls in Certain Columns 34](#_Toc521589553)

[Delete Duplicate Rows 34](#_Toc521589554)

[Display DataFrame column types 34](#_Toc521589555)

[Extract a column from a DataFrame into a Series 34](#_Toc521589556)

[Get the rows in a DataFrame having a null in some column 35](#_Toc521589557)

[Fast update of a DataFrame column 35](#_Toc521589558)

[Filter out na values from a column 35](#_Toc521589559)

[Find the Row Index in a DataFrame with a Particular Value 36](#_Toc521589560)

[Row Index Matching Values in Multiple Columns 36](#_Toc521589561)

[Get DataFrame column names 36](#_Toc521589562)

[Get DataFrame column values 36](#_Toc521589563)

[Get dimensions of a DataFrame 37](#_Toc521589564)

[Get row count from a DataFrame 37](#_Toc521589565)

[Get rows from a DataFrame by index 37](#_Toc521589566)

[Get Rows from a DataFrame which Match an Element of a List 37](#_Toc521589567)

[Get unique values from a DataFrame column 38](#_Toc521589568)

[GroupBy Functionality 39](#_Toc521589569)

[Extract the values from a GroupBy 39](#_Toc521589570)

[Extract only the rows containing the max value of a column, in a GroupBy 40](#_Toc521589571)

[Insert a column into a DataFrame 40](#_Toc521589572)

[Keep Only Certain Columns of a DataFrame 40](#_Toc521589573)

[Max value of a DataFrame column 40](#_Toc521589574)

[Plot the Data in a DataFrame 41](#_Toc521589575)

[Scatter Plot 41](#_Toc521589576)

[Randomly Split a DataFrame 42](#_Toc521589577)

[Random Sample from a DataFrame 43](#_Toc521589578)

[Read a CSV file into a DataFrame 43](#_Toc521589579)

[Re-Sample a DataFrame to Aggregate 43](#_Toc521589580)

[Parse formatted dates while reading a CSV 43](#_Toc521589581)

[Remove Rows which Match Elements of a List 44](#_Toc521589582)

[Reset DataFrame Data Type 44](#_Toc521589583)

[Reset DataFrame Indices 44](#_Toc521589584)

[Sample a DataFrame 44](#_Toc521589585)

[Scale Multiple Columns of a DataFrame 45](#_Toc521589586)

[Select a cell from a DataFrame 45](#_Toc521589587)

[Select rows from a DataFrame by value of a column 45](#_Toc521589588)

[Select rows from a DataFrame by values of multiple columns 46](#_Toc521589589)

[Select rows having NaN or null in Multiple Columns 46](#_Toc521589590)

[Sort a DataFrame 47](#_Toc521589591)

[Sort by a column 47](#_Toc521589592)

[Sort by multiple columns 47](#_Toc521589593)

[Sort Descending 47](#_Toc521589594)

[Split a DataFrame into train and test sets 48](#_Toc521589595)

[Split a DataFrame into train, validate, and test sets 48](#_Toc521589596)

[Substitute for na values in a column 48](#_Toc521589597)

[Summary statistics for a DataFrame 49](#_Toc521589598)

[Write a DataFrame to a csv file 49](#_Toc521589599)

[Wrapping CSV file columns in quotes 49](#_Toc521589600)

[Date Functions 49](#_Toc521589601)

[Add a time interval to a datetime 49](#_Toc521589602)

[Calculate a time interval 50](#_Toc521589603)

[Calculate a time interval in seconds, days 50](#_Toc521589604)

[Convert a datetime to Epoch Seconds 51](#_Toc521589605)

[Convert an Epoch to a time 51](#_Toc521589606)

[Convert string to date 51](#_Toc521589607)

[Microseconds 51](#_Toc521589608)

[Date Time Format Strings 52](#_Toc521589609)

[Another method: 52](#_Toc521589610)

[Create an arbitrary datetime 52](#_Toc521589611)

[datetime with time zone 53](#_Toc521589612)

[Get the current datetime 53](#_Toc521589613)

[Get year, month, day, hour, minute, second, milliseconds, weekday 53](#_Toc521589614)

[ISO Weekday 54](#_Toc521589615)

[Time Zone Names 54](#_Toc521589616)

[Dictionaries 54](#_Toc521589617)

[Convert a DataFrame to a Dictionary 54](#_Toc521589618)

[Create a dictionary 54](#_Toc521589619)

[Execute a Function on all the Values in a Dictionary 54](#_Toc521589620)

[Get a value for a key in the dict 55](#_Toc521589621)

[Get the keys from a dictionary 55](#_Toc521589622)

[Is a key in a dictionary? 55](#_Toc521589623)

[Directories 55](#_Toc521589624)

[Check if a Directory exists 55](#_Toc521589625)

[Concatenate a Directory and File Name 56](#_Toc521589626)

[Create a Directory 56](#_Toc521589627)

[Delete all the files and folders in a directory 56](#_Toc521589628)

[Delete all the files in a directory 56](#_Toc521589629)

[Get the Current Working Directory 57](#_Toc521589630)

[Read the files in a directory. 57](#_Toc521589631)

[Read the files in a directory with a specific extension 57](#_Toc521589632)

[Set the working directory 57](#_Toc521589633)

[Exception Handling 57](#_Toc521589634)

[try-except 57](#_Toc521589635)

[Print the traceback and stack trace 58](#_Toc521589636)

[Files 58](#_Toc521589637)

[Copy a file between from one directory to another 58](#_Toc521589638)

[Copy a File from a URL 58](#_Toc521589639)

[Delete a file 58](#_Toc521589640)

[Does a file exist? 58](#_Toc521589641)

[Extract the Filename and Extension from a path 59](#_Toc521589642)

[Extract the file name from a path 59](#_Toc521589643)

[Open File dialog 59](#_Toc521589644)

[Read a text file into a string 59](#_Toc521589645)

[Read all the lines in a file into a list 59](#_Toc521589646)

[Read a text file line by line 60](#_Toc521589647)

[Read a CSV file 60](#_Toc521589648)

[Write to a Text File 61](#_Toc521589649)

[Geocoding 61](#_Toc521589650)

[Geography 62](#_Toc521589651)

[Distance between two coordinates 62](#_Toc521589652)

[Hash Functions 62](#_Toc521589653)

[Images 63](#_Toc521589654)

[View an Image using matplotlib 63](#_Toc521589655)

[Installing packages 63](#_Toc521589656)

[easy\_install 63](#_Toc521589657)

[json 64](#_Toc521589658)

[Pretty Print JSON 64](#_Toc521589659)

[Reading a json file into a dict 64](#_Toc521589660)

[Jupyter Notebooks 64](#_Toc521589661)

[Display an Image inside a Notebook 64](#_Toc521589662)

[Display matplotlib plots inline in the notebook 64](#_Toc521589663)

[Store a CSV file in the local directory (not the HDFS directory) 64](#_Toc521589664)

[Lambdas 65](#_Toc521589665)

[Conditional Lambdas 65](#_Toc521589666)

[Libraries 65](#_Toc521589667)

[Find the Function Available in a Library 65](#_Toc521589668)

[Lists 66](#_Toc521589669)

[Apply Functions to the Elements of a List 66](#_Toc521589670)

[Average of items in a list 66](#_Toc521589671)

[Cartesian Product of 2 Lists 66](#_Toc521589672)

[Concatenate 2 lists 66](#_Toc521589673)

[Concatenate the string elements of 2 Lists 66](#_Toc521589674)

[Copy a list 66](#_Toc521589675)

[Create a list containing a number of constants 67](#_Toc521589676)

[Convert a list to a dict 67](#_Toc521589677)

[Count the Number of Occurences of an Item in a List 67](#_Toc521589678)

[Creating and Appending to a List 67](#_Toc521589679)

[Filter a List 67](#_Toc521589680)

[Last items in a List 68](#_Toc521589681)

[Merge 2 Lists with Option to Remove Dupes 68](#_Toc521589682)

[Randomly Split a List 68](#_Toc521589683)

[Randomly Sample Items from a List 68](#_Toc521589684)

[Randomly Sample Items from a List with Replacement 69](#_Toc521589685)

[Remove an Item from a List 69](#_Toc521589686)

[Remove Null Values from a List 69](#_Toc521589687)

[Replace an item in a list 69](#_Toc521589688)

[Sort a list 69](#_Toc521589689)

[Shuffle the items in a list 70](#_Toc521589690)

[Subtract the Elements in 2 Lists 70](#_Toc521589691)

[Standard Deviation of items in a list 70](#_Toc521589692)

[Using lambda and map on a list 70](#_Toc521589693)

[Machine Learning 70](#_Toc521589694)

[Create Word Count columns 70](#_Toc521589695)

[Euclidean Distance 70](#_Toc521589696)

[One-Hot Encoder 71](#_Toc521589697)

[Maps 72](#_Toc521589698)

[folium to easily create a map 72](#_Toc521589699)

[Math Functions 72](#_Toc521589700)

[Exponentiation 72](#_Toc521589701)

[Largest float 73](#_Toc521589702)

[Median 73](#_Toc521589703)

[Modulo 73](#_Toc521589704)

[pi 73](#_Toc521589705)

[Random Numbers 73](#_Toc521589706)

[Random float 73](#_Toc521589707)

[Set the Random Number Seed 73](#_Toc521589708)

[import random 73](#_Toc521589709)

[Rounding 74](#_Toc521589710)

[General rounding 74](#_Toc521589711)

[Round to half-even 74](#_Toc521589712)

[Round to {x.0, x.5} intervals 75](#_Toc521589713)

[Scale a 2-Dimensional List 75](#_Toc521589714)

[Square Root 76](#_Toc521589715)

[Test for nan 76](#_Toc521589716)

[Matrices 76](#_Toc521589717)

[Number of rows in a matrix 76](#_Toc521589718)

[Read a Matrix from a file 76](#_Toc521589719)

[Read the contents of a matrix column into an array 76](#_Toc521589720)

[Scale matrix columns 76](#_Toc521589721)

[Methods 77](#_Toc521589722)

[Method Header Template 77](#_Toc521589723)

[numpy 77](#_Toc521589724)

[Covariance 77](#_Toc521589725)

[Element-Wise Multiplication of two Arrays or Iterables 77](#_Toc521589726)

[r-squared 77](#_Toc521589727)

[Variance 78](#_Toc521589728)

[Object Serialization 78](#_Toc521589729)

[Create an object from a stored serialization 78](#_Toc521589730)

[Store and then Recall a Binary Object 78](#_Toc521589731)

[Serialize and Store an Object 79](#_Toc521589732)

[Simple Example - Save and Recall a dictionary 79](#_Toc521589733)

[Store a Binary Object 79](#_Toc521589734)

[Packages 79](#_Toc521589735)

[Check Package Version 79](#_Toc521589736)

[pandas 79](#_Toc521589737)

[Change the number of rows printed for pandas objects 79](#_Toc521589738)

[pandasql 80](#_Toc521589739)

[Installing pandasql 80](#_Toc521589740)

[Querying using pandasql 80](#_Toc521589741)

[Set the max number of columns displayed when showing a DataFrame 80](#_Toc521589742)

[Plotting 81](#_Toc521589743)

[Histograms 81](#_Toc521589744)

[Line + Scatter Plots using plotly 81](#_Toc521589745)

[Scatter plot 82](#_Toc521589746)

[Program Execution 82](#_Toc521589747)

[Stopping program execution 83](#_Toc521589748)

[pyspark 83](#_Toc521589749)

[pyspark.sql.dataframe.DataFrame 83](#_Toc521589750)

[Column Names 83](#_Toc521589751)

[Extract a List from a pyspark.sql.dataframe.DataFrame Column 83](#_Toc521589752)

[Read in a file from the server’s local OS (not HDFS) 83](#_Toc521589753)

[Show the values in an array column 83](#_Toc521589754)

[pyspark version 84](#_Toc521589755)

[Write a pySpark DataFrame to a CSV File 84](#_Toc521589756)

[Python version 84](#_Toc521589757)

[Regular expressions 84](#_Toc521589758)

[Remove punctuation 84](#_Toc521589759)

[Random Numbers 84](#_Toc521589760)

[Random Integer in a Range 84](#_Toc521589761)

[Choose Random Items from a List 84](#_Toc521589762)

[Create a list containing some random numbers 85](#_Toc521589763)

[REST Services 85](#_Toc521589764)

[Consume a REST service 85](#_Toc521589765)

[Consume an XML Service 85](#_Toc521589766)

[scikit-learn 85](#_Toc521589767)

[GroupKFold Implementation for Cross-Validation 85](#_Toc521589768)

[Linear regression 86](#_Toc521589769)

[sklearn Version 86](#_Toc521589770)

[Check if a Series value is null 87](#_Toc521589771)

[Convert a Series to a DataFrame 87](#_Toc521589772)

[Create a Series of random numbers 87](#_Toc521589773)

[Get the value of a Series element 87](#_Toc521589774)

[SFrame 87](#_Toc521589775)

[Add a Column Based on Other Columns 87](#_Toc521589776)

[Convert an SFrame to features and labels in a numpy array 87](#_Toc521589777)

[Copy an Sframe 88](#_Toc521589778)

[First n rows of an Sframe 88](#_Toc521589779)

[One-Hot Encoding of an Sframe 88](#_Toc521589780)

[Random Split an SFrame 89](#_Toc521589781)

[Remove a Column from an Sframe 89](#_Toc521589782)

[Select Rows from an Sframe 89](#_Toc521589783)

[Statistics 89](#_Toc521589784)

[Applying lowess smoothing 89](#_Toc521589785)

[Precision, recall, F1, support 90](#_Toc521589786)

[Strings 90](#_Toc521589787)

[Concatenate strings 90](#_Toc521589788)

[Convert a character to its ASCII integer 90](#_Toc521589789)

[Convert to float 90](#_Toc521589790)

[Convert to lower case 90](#_Toc521589791)

[Find a sub-string 91](#_Toc521589792)

[Find nth Occurrence of a sub-string 91](#_Toc521589793)

[Formatted strings 91](#_Toc521589794)

[Right-Pad a Numeric Formatted String with zeros 91](#_Toc521589795)

[Remove Punctuation 91](#_Toc521589796)

[Replace a substring 92](#_Toc521589797)

[String Literals 92](#_Toc521589798)

[Sub-strings 92](#_Toc521589799)

[Tokenize a string 92](#_Toc521589800)

[Trim leading and trailing characters 92](#_Toc521589801)

[Trim white space 92](#_Toc521589802)

[Timers 93](#_Toc521589803)

[Sleep 93](#_Toc521589804)

[Timing Code Execution 93](#_Toc521589805)

[Tuples 93](#_Toc521589806)

[Cartesion product of two tuples 93](#_Toc521589807)

[Product of the elements in a tuple 93](#_Toc521589808)

[Verify 2 tuples contain the same elements 94](#_Toc521589809)

[User Input 94](#_Toc521589810)

[Get user input from the keyboard 94](#_Toc521589811)

[XML 94](#_Toc521589812)

[Errors 95](#_Toc521589813)

# Arrays (numpy)

## Add a column to an array

>>> N = 3

>>> A = np.eye(N)

>>> A

array([[ 1., 0., 0.],

[ 0., 1., 0.],

[ 0., 0., 1.]])

>>> np.c\_[A, np.zeros(N)]

array([[ 1., 0., 0., 0.],

[ 0., 1., 0., 0.],

[ 0., 0., 1., 0.]])

>>>

## Add a dimension to an array

>>> import numpy as np

>>> x = np.array([1,2])

>>> x

array([1, 2])

>>> x.shape

(2L,)

>>> y = np.expand\_dims(x, axis=0)

>>> y

array([[1, 2]])

>>> y.shape

(1L, 2L)

## Apply a function to every element of an array

get\_pred = np.vectorize(lambda x: 1 if x > 0 else -1)

y\_hat = get\_pred(scores) # scores is type numpy.ndarray

print y\_hat

output:

[ 1 -1 1 ..., -1 1 -1]

## Change the Data Type of an Array

>>> x = np.array([1, 2, 2.5])

>>> x

array([ 1. , 2. , 2.5])

>>> x.astype(int)

array([1, 2, 2])

## Concatenate Arrays

>>> a = np.array([[1, 2], [3, 4]])

>>> b = np.array([[5, 6]])

>>> np.concatenate((a, b), axis=0)

array([[1, 2],

[3, 4],

[5, 6]])

>>> np.concatenate((a, b.T), axis=1)

array([[1, 2, 5],

[3, 4, 6]])

## Convert a DataFrame to an Array

input\_nda = corr\_input\_df.values

## Convert a list to an array

>>> blist = [2,4,6,8]

>>> b\_array = np.array(blist)

>>> b\_array

array([2, 4, 6, 8])

## Count the number of equal items in two arrays

import numpy as np

a = np.array([1, 2, 3, 4])

b = np.array([1, 2, 4, 3])

np.sum(a == b) # (a == b) returns [True, True, False, False], but

# np.sum() interprets the Trues as 1

output:

2

(a == b).sum()

output:

2

## Create an array of constants

import numpy as np

a = np.empty(10)

a.fill(55)

## Create an array of zeros

>>> import numpy as np

>>> np.zeros(10)

array([ 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])

### 2D

>>> test = np.zeros((10,10))

>>> test

array([[ 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],

[ 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],

[ 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],

[ 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],

[ 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],

[ 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],

[ 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],

[ 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],

[ 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],

[ 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]])

## Create and add rows to an array

import numpy as np

arr = np.empty((0,3), float) # arr is type numpy.ndarray

arr = np.append(arr, np.array([[1,2,3]]), axis=0)

arr = np.append(arr, np.array([[4,5,6]]), axis=0)

print arr

>>>

[[ 1. 2. 3.]

[ 4. 5. 6.]]

>>>

## Create a 2-D Array From Scratch

test\_dist\_129 = np.ndarray(shape=(2,3),

buffer = np.array([[1.0,2.0,3.0],

[4.0,5.0,6.0]]),

dtype = float)

print test\_dist\_129

output:

[[ 1. 2. 3.]

[ 4. 5. 6.]]

<type 'numpy.ndarray'>

## Dimensions of an array

>>> X\_train.shape

(80L, 5L)

## Dot Product of 2 arrays

u = np.arange(0, 5, .5)

v = np.arange(5, 10, .5)

u: [ 0. 0.5 1. 1.5 2. 2.5 3. 3.5 4. 4.5]

v: [ 5. 5.5 6. 6.5 7. 7.5 8. 8.5 9. 9.5]

dotProduct = np.dot(u,v)

dotProduct

183.75

You can also do this for a matrix and array.

Note that numpy will automatically transpose the second argument, if needed, to make the two arguments consistent.

## Element-wise Multiplication

u = np.arange(0, 5, .5)

v = np.arange(5, 10, .5)

elementWise = u \* v

u: [ 0. 0.5 1. 1.5 2. 2.5 3. 3.5 4. 4.5]

v: [ 5. 5.5 6. 6.5 7. 7.5 8. 8.5 9. 9.5]

elementWise

[ 0. 2.75 6. 9.75 14. 18.75 24. 29.75 36. 42.75]

## Expand the Rank (number of dimensions) of an array

>>> import numpy as np

>>> x = np.array([1,2])

>>> x

array([1, 2])

>>> x.shape

(2L,)

>>> y = np.expand\_dims(x, axis=0)

>>> y

array([[1, 2]])

>>> y.shape

(1L, 2L)

>>> len(x.shape)

1

>>> len(y.shape)

2

## Flatten a numpy Array

>>> a = np.array([[1,2], [3,4]])

>>> a.flatten()

array([1, 2, 3, 4])

>>> a.flatten('F')

array([1, 3, 2, 4])

## Load a CSV file into a numpy array

from numpy import genfromtext

filepath = "eta\_corr\_owner\_veh\_km-rem\_mins-rem\_pace\_lat\_lon\_hr\_day\_mo\_no-dupe.csv"

my\_data = genfromtxt(filepath, delimiter=',', skip\_header=1) # assumes 1 header row

Normalize Columns of an ndarray

## Normalize Rows of an ndarray

print non\_zero\_dist\_array

weights\_array = normalize(non\_zero\_dist\_array, axis=1, norm='l1')

print weights\_array

Output:

[[ 1. 2. 3.]

[ 4. 5. 6.]]

[[ 0.16666667 0.33333333 0.5 ]

[ 0.26666667 0.33333333 0.4 ]]

## print() options

### print entire array

np.set\_printoptions(threshold=np.inf)

## Reshape an array

Reshape an array into a Single Row or Column

import numpy as np

X = np.array((3,4))

y = X.reshape(1, -1) # Create an array of rows

z = X.reshape(-1, 1) # Create an array of columns

print "X=", X

print "y=", y

print "z=", z

Output:

X= [3 4]

y= [[3 4]]

z= [[3]

[4]]

## Select a Subset of Columns

X = input\_nda[:, [6, 9, 10, 11, 12, 14, 15]] # Selects columns 6, 9, …

## Select a Subset of Rows

import numpy as np

data = np.array([

[100002, 2006, 1.1, 0.01, 6352],

[100002, 2006, 1.2, 0.84, 304518],

[100002, 2006, 2, 1.52, 148219],

[100002, 2007, 1.1, 0.01, 6292],

[10002, 2006, 1.1, 0.01, 5968],

[10002, 2006, 1.2, 0.25, 104318],

[10002, 2007, 1.1, 0.01, 6800],

[10002, 2007, 4, 2.03, 25446],

[10002, 2008, 1.1, 0.01, 6408] ])

subset1 = data[data[:,0] == 100002]

subset1:

array([[ 1.00002e+05, 2.006e+03, 1.10e+00, 1.00e-02, 6.352e+03],

[ 1.00002e+05, 2.006e+03, 1.20e+00, 8.40e-01, 3.04518e+05],

[ 1.00002e+05, 2.006e+03, 2.00e+00, 1.52e+00, 1.48219e+05],

[ 1.00002e+05, 2.007e+03, 1.10e+00, 1.00e-02, 6.292e+03]])

## Select Rows from an Array which Match a List of Values for some Column

import numpy as np

training\_shipments, test\_shipments = train\_test\_split(unique\_shipment\_id\_array,

test\_size=0.3,

random\_state=0)

print all\_data.shape

training\_data = all\_data[np.logical\_or.reduce([all\_data[:,9] == x

for x in training\_shipments])]

print my\_data.shape

(63L, 18L)

(53L, 18L)

## Shuffle an Array

Note that both these code segments shuffle a feature and target array, in unison.

idx = np.arange(X.shape[0])

np.random.seed(13)

np.random.shuffle(idx)

X = X[idx]

y = y[idx]

or

# Shuffle the data before starting

permutation = np.random.permutation(len(feature\_matrix))

feature\_matrix = feature\_matrix[permutation,:]

sentiment = sentiment[permutation]

## Slicing an Array

### Return the last 3 elements

features = np.array([1, 2, 3, 4])

print 'features:\n{0}'.format(features)

# The last three elements of features

lastThree = features[-3:]

print '\nlastThree:\n{0}'.format(lastThree)

features:

[1 2 3 4]

lastThree:

[2 3 4]

## Standardize an Array

mean = X.mean(axis=0)

std = X.std(axis=0)

X = (X - mean) / std

## Statistics for Arrays

### Median for Arrays

>>> a

array([[10, 7, 4],

[ 3, 2, 1]])

>>> np.median(a)

3.5

>>> np.median(a, axis=0)

array([ 6.5, 4.5, 2.5])

>>> np.median(a, axis=1)

array([ 7., 2.])

## Sort a numpy.ndarray

>>> unique\_dates

array(['2014-10-21', '2014-10-11', '2014-10-14', '2014-10-15',

'2014-10-16', '2014-10-17', '2014-10-18', '2014-10-20',

'2014-10-06', '2014-10-07', '2014-10-09', '2014-10-10',

'2014-10-08', '2014-09-26', '2014-10-13', '2014-10-12',

'2014-10-05', '2014-10-19', '2014-09-30', '2014-10-03',

'2014-10-01', '2014-10-02'], dtype=object)

>>> import numpy as np

>>> np.sort(unique\_dates)

array(['2014-09-26', '2014-09-30', '2014-10-01', '2014-10-02',

'2014-10-03', '2014-10-05', '2014-10-06', '2014-10-07',

'2014-10-08', '2014-10-09', '2014-10-10', '2014-10-11',

'2014-10-12', '2014-10-13', '2014-10-14', '2014-10-15',

'2014-10-16', '2014-10-17', '2014-10-18', '2014-10-19',

'2014-10-20', '2014-10-21'], dtype=object)

## Split an Array into 2 parts

import numpy as np

x = np.arange(9)

train = x[:4]

print train

val = x[4:]

print val

>>>

[0 1 2 3 4 5 6 7 8]

>>> ================================ RESTART ================================

[0 1 2 3]

[4 5 6 7 8]

## Transpose a numpy.ndarray

>>> a = np.array([[1, 2], [3, 4]])

>>> a

array([[1, 2],

[3, 4]])

>>> a.transpose()

array([[1, 3],

[2, 4]])

# Classes

class Dog:

def \_\_init\_\_(self, name):

self.name = name

self.tricks = [] # creates a new empty list for each dog

def add\_trick(self, trick):

self.tricks.append(trick)

>>> d = Dog('Fido')

>>> e = Dog('Buddy')

>>> d.add\_trick('roll over')

>>> e.add\_trick('play dead')

>>> d.tricks

['roll over']

>>> e.tricks

['play dead']

## Class static methods

""" CongregExtractor: Contains the CongregExtractor class """

class CongregExtractor:

""" CongregExtractor class """

def \_\_init\_\_(self):

self.congreg = []

@staticmethod

def stops\_to\_congreg(stop\_list):

congreg\_list = ["list"]

return(congreg\_list)

Invoke like:

from CongregExtractor import CongregExtractor as ce

my\_congregations = ce.stops\_to\_congreg(my\_items\_list)

# Control Statements

## for-next loops

>>> for x in xrange(0,3):

print(x)

break # causes immediate termination of the loop

0

1

2

### Count backwards

for i in xrange(100,-1,-1)

### for-next over multiple variables

import itertools

for i,j in itertools.product(range(3), range(2)):

print i,j

>>>

0 0

0 1

1 0

1 1

2 0

2 1

## while loops

count = 0

while (count < 9):

print 'The count is:', count

count = count + 1

print "Good bye!"

# CSV Files

## Read from a CSV file

import csv

with open(‘test\_tag\_speed\_km\_mins\_pace\_hour\_day\_6mos\_end\_2012\_05\_22.csv', 'rb') as csvfile:

myreader = csv.reader(csvfile, delimiter=',')

for row in myreader:

print(row) # each row is a list containing the fields

>>> df = pd.read\_csv('outlier\_data/journey\_owner\_duration\_end\_date\_with\_history.csv')

>>> df.shape

(14410, 6)

>>> type(df)

<class 'pandas.core.frame.DataFrame'>

## Write to a CSV file

import csv

with open('eggs.csv', 'wb') as csvfile:

spamwriter = csv.writer(csvfile, delimiter=' ',

quotechar='|', quoting=csv.QUOTE\_MINIMAL)

spamwriter.writerow(['Spam'] \* 5 + ['Baked Beans'])

spamwriter.writerow(['Spam', 'Lovely Spam', 'Wonderful Spam'])

# Database Functions

## Postgres

import psycopg2

import psycopg2.extensions

### Connect to a Postgres database

def get\_db\_conn():

conn = None

try:

conn = psycopg2.connect("dbname='dbanalytics' " +

"user='analytics' " +

"host='analytics.ca9tgiacgkgn.us-west-2.rds.amazonaws.com' " +

"password='<password-here>'")

print "Connected to dbanalytics database"

except:

print "Unable to connect to database"

return(conn)

### Execute SQL query on Postgres

conn = get\_db\_conn()

cur = conn.cursor()

with open('UC1\_query.sql', 'r') as uc1\_query\_file:

uc1\_query = uc1\_query\_file.read()

cur.execute(uc1\_query)

test\_output = cur.fetchall()

df = pd.DataFrame(test\_output)

df.columns = ['sap\_shipment\_id', 'status\_code', 'status\_reason',

'status\_dt\_utc', 'status\_lat', 'status\_lng',

'sap\_shipping\_point', 'orig\_lat', 'orig\_lon',

'dest\_lat', 'dest\_lon', 'driver\_eta\_utc',

'lat\_lon\_corr\_names', 'sap\_ship\_so',

'planned\_shipment\_end\_date\_utc\_c',

'actual\_shipment\_end\_date\_utc',

'sap\_message\_date\_utc\_c']

## SQLite3

### Create a SQLite3 Database

import sqlite3

conn = sqlite3.connect("lat\_lon\_db")

cursor = conn.cursor()

cursor.execute("create table address(name text PRIMARY\_KEY, latitude real, longitude real)")

conn.commit()

conn.close()

## Data Types

Each value stored in an SQLite database (or manipulated by the database engine) has one of the following storage classes:

•NULL. The value is a NULL value.

•INTEGER. The value is a signed integer, stored in 1, 2, 3, 4, 6, or 8 bytes depending on the magnitude of the value.

•REAL. The value is a floating point value, stored as an 8-byte IEEE floating point number.

•TEXT. The value is a text string, stored using the database encoding (UTF-8, UTF-16BE or UTF-16LE).

•BLOB. The value is a blob of data, stored exactly as it was input.

### Insert Values into Database

import sqlite3

conn = sqlite3.connect("lat\_lon\_db")

conn.text\_factory = lambda x: unicode(x, 'utf-8', 'ignore')

cursor = conn.cursor()

cursor.execute("insert into address (name, latitude, longitude) values (?,?,?)",

("4704 Nelson Brogdon Blvd Ne Sugar Hill GA 30518", 34.1025827, -84.02279469999999))

conn.commit()

conn.close()

Note: failure to use the above text\_factory setting has resulted in errors like: “ProgrammingError: You must not use 8-bit bytestrings unless you use a text\_factory that can interpret 8-bit bytestrings (like text\_factory = str). It is highly recommended that you instead just switch your application to Unicode strings.”

### Read from a Database Table

import sqlite3

conn = sqlite3.connect("lat\_lon\_db")

cursor = conn.cursor()

result = cursor.execute("select \* from address")

print(result.fetchall())

### Parameterized Queries

import sqlite3

conn = sqlite3.connect("lat\_lon\_db")

cursor = conn.cursor()

result = cursor.execute("select \* from address where street\_address = ?", ['1439 Buckeye Court Auburn CA 95603'])

x = result.fetchall()

conn.close()

# DataFrame (pandas)

## Add a column to a DataFrame

from pandas import DataFrame

>>> loc\_data\_frame.shape

(86, 11)

>>> loc\_data\_frame['predict'] = np.zeros(86)

>>> loc\_data\_frame.shape

(86, 12)

-or-

>>> from pandas import DataFrame

>>> test=DataFrame()

>>> my\_list = [1,2,3]

>>> test['col\_name'] = my\_list

>>> test

col\_name

0 1

1 2

2 3

-or-

from pandas import DataFrame

import numpy as np

test=DataFrame()

my\_list = [1,2,3]

test['number'] = my\_list

test['squared'] = test['number'] \*\*2

test['is\_even'] = np.where(test.squared % 2 == 0, True, False)

test['is\_odd'] = test.apply(lambda row: (row.squared % 2 == 1), axis=1)

print test

>>>

number squared is\_even is\_odd

0 1 1 False True

1 2 4 True False

2 3 9 False True

## Add a row to a DataFrame

>>> mydf = DataFrame(columns=("A", "B"))

>>> mydf

Empty DataFrame

Columns: [A, B]

Index: []

>>> mydf.loc[len(mydf)] = [18,19]

>>> mydf

A B

0 18 19

df637 = DataFrame(columns=('status\_dt\_utc',))

df637.loc[0] = [datetime(2016, 4, 27, 12, 0, 0)]

df637.loc[1,['status\_dt\_utc']] = datetime(2016, 4, 27, 12, 1, 0)

print df637

output:

status\_dt\_utc

0 2016-04-27 12:00:00

1 2016-04-27 12:01:00

## Apply a Lambda to Every Row of a DataFrame

shipment\_df['sec\_diff'] = shipment\_df.apply(lambda row: (row['planned\_arrival\_utc'] -

row['status\_dt\_utc']).total\_seconds(),

axis = 1)

## Apply a function, with arguments

df958 = DataFrame(columns=("SOME\_LAT", "SOME\_LON"))

df958.loc[0] = [38.896, -121.077] # Auburn, CA, in LA time zone

df958['TZ'] = df958.apply(get\_tz, args=('SOME\_LAT', 'SOME\_LON'), axis=1)

assert df958.TZ[0] == 'America/Los\_Angeles'

**IMPORTANT: If you only have a single argument, you must encode it like:**

df531['at\_orig'] = df531.apply(get\_at\_orig\_flag, args=(poi\_df\_531,),

axis=1)

## Change column names

df = pd.DataFrame(test\_output)

df.columns = ['sap\_shipment\_id', 'status\_code']

-or-

import pandas as pd

df = pd.DataFrame({"A": [1, 2, 3], "B": [4, 5, 6]})

print df

df = df.rename(columns={"A": "a", "B": "c"})

print df

A B

0 1 4

1 2 5

2 3 6

a c

0 1 4

1 2 5

2 3 6

## Change Column Data Type

a = [['a', '1.2', '4.2'], ['b', '70', '0.03'], ['x', '5', '0']]

df = pd.DataFrame(a, columns=['one', 'two', 'three'])

df

Out[16]:

one two three

0 a 1.2 4.2

1 b 70 0.03

2 x 5 0

df.dtypes

Out[17]:

one object

two object

three object

df[['two', 'three']] = df[['two', 'three']].astype(float)

df.dtypes

Out[19]:

one object

two float64

three float64

## Change values in one column based on values in a different column

from pandas import DataFrame, read\_csv, Series

df = DataFrame(columns=('A', 'B', 'C'))

df.loc[0] = [1,2,3]

df.loc[1] = [1,2,4]

df.loc[2] = [1,7,8]

print df

df.loc[df['B'] == 2, 'C'] = 13

print df

>>>

2015-08-31 03:22:56,512 tzwhere.py <module> 42 INFO Application started..

A B C

0 1 2 3

1 1 2 4

2 1 7 8

A B C

0 1 2 13

1 1 2 13

2 1 7 8

## Concatenate two data frames

from pandas import concat

>>> df1

A B C

0 1 2 3

1 3 2 1

>>> df2

D E F

0 5 6 7

1 7 6 5

>>> concat([df1, df2], axis=1)

A B C D E F

0 1 2 3 5 6 7

1 3 2 1 7 6 5

## Convert a DataFrame Column to type datetime

import pandas as pd

ship\_df[['actual\_arrival\_utc']] = pd.to\_datetime(ship\_df['actual\_arrival\_utc'])

## Convert a Single DataFrame Column to a numpy.ndarray

X\_df = trimmed\_model\_input\_df[['leg\_km\_to\_port\_boundary']]

Output:

leg\_km\_to\_port\_boundary

0 3.336571

1 8836.782944

2 2234.021963

3 0.000000

4 7.770299

<class 'pandas.core.frame.DataFrame'>

leg\_km\_arr = np.reshape(X\_df.leg\_km\_to\_port\_boundary, (len(X\_df.leg\_km\_to\_port\_boundary), 1))

print type(leg\_km\_arr)

print leg\_km\_arr

Output:

<type 'numpy.ndarray'>

[[ 3.33657117e+00]

[ 8.83678294e+03]

[ 2.23402196e+03]

...,

[ 1.38211981e+03]

[ 1.77436680e+01]

[ 1.80216301e+03]]

## Convert a DataFrame to a numpy.ndarray

from pandas import DataFrame

df = DataFrame(columns=('A', 'B', 'C'))

df.loc[0] = [1, 2, 3]

df.loc[1] = [4, 5, 6]

print df

v = df.values

print v

print type(v)

Output:

>>>

A B C

0 1 2 3

1 4 5 6

[[ 1. 2. 3.]

[ 4. 5. 6.]]

<type 'numpy.ndarray'>

## Convert a Dict to a DataFrame

prod\_fname = r"algProd.out" # JSON

with open(prod\_fname) as prod\_file:

prod\_dict = json.load(prod\_file)

prod\_pred\_df = DataFrame.from\_dict(prod\_dict['response']['docs'])

list(prod\_pred\_df)

Output:

[u'\_uniqueKey',

u'features',

u'features\_num',

u'mkey',

u'model\_category',

u'model\_class',

u'model\_err',

u'model\_md5',

u'model\_type',

u'obsv\_asof',

u'strata\_carrier',

u'strata\_num',

u'strata\_odpair',

u'strata\_shipper']

## Copy a column from another DataFrame

# note input\_df will be a DataFrame

input\_df = read\_csv(input\_fname, header=0, quotechar='"',

quoting=csv.QUOTE\_ALL )

corr\_names = DataFrame(input\_df, columns=['CORRIDOR'])

## Copy a DataFrame

DataFrame.**copy**(deep=True)[¶](http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.copy.html#pandas.DataFrame.copy)

Make a copy of this objects data.

|  |  |
| --- | --- |
| **Parameters:** | **deep** : boolean or string, default True  Make a deep copy, including a copy of the data and the indices. With deep=False neither the indices or the data are copied.  Note that when deep=True data is copied, actual python objects will not be copied recursively, only the reference to the object. This is in contrast to copy.deepcopy in the Standard Library, which recursively copies object data. |
| **Returns:** | **copy** : type of caller |

comparison\_df = prod\_cv\_score\_df.copy(deep=True)

## Correlation between columns

import pandas as pd

import matplotlib.pylab as plt

df = pd.read\_csv('winequality-red.csv', sep=';')

print(df.corr())

## Count the distinct values in a DataFrame column

from pandas import DataFrame, Series

import pandas as pd; import numpy as np

frame = DataFrame(records)

frame['tz'].value\_counts()

America/New\_York 1251

521

America/Chicago 400

America/Los\_Angeles 382

America/Denver 191

Europe/London 74

Asia/Tokyo 37

Pacific/Honolulu 36

Europe/Madrid 35

America/Sao\_Paulo 33

Europe/Berlin 28

Europe/Rome 27

America/Rainy\_River 25

Europe/Amsterdam 22

America/Indianapolis 20

...

Europe/Ljubljana 1

Asia/Riyadh 1

## Create a DataFrame from scratch

from pandas import DataFrame

df = DataFrame(columns=('A', 'B', 'C'))

df

Empty DataFrame

Columns: [A, B, C]

Index: []

>>> df.loc[0] = [1, 2, 3]

>>> df

A B C

0 1 2 3

- OR –

import pandas as pd

import numpy as np

import datetime as dt

df2 = pd.DataFrame({ 'A' : 1.,

'B' : pd.Timestamp('20130102'),

'C' : pd.Series(1,index=list(range(4)),dtype='float32'),

'D' : np.array([3] \* 4,dtype='int32'),

'E' : pd.Categorical(["test","train","test","train"]),

'F' : 'foo',

'G' : ['A', 'B', 'C', 'D'],

'H' : [dt.datetime(2016,9,5,1,2,3),

dt.datetime(2016,9,5,2,3,4),

dt.datetime(2016,9,5,3,4,5),

dt.datetime(2016,9,5,4,5,6)]})

print df2

>>>

A B C D E F G H

0 1 2013-01-02 1 3 test foo A 2016-09-05 01:02:03

1 1 2013-01-02 1 3 train foo B 2016-09-05 02:03:04

2 1 2013-01-02 1 3 test foo C 2016-09-05 03:04:05

3 1 2013-01-02 1 3 train foo D 2016-09-05 04:05:06

### Create a DataFrame which has only one column

>>> df = DataFrame(columns=('A',))

>>> df

Empty DataFrame

Columns: [A]

Index: []

## Create a DataFrame with integer columns

test002\_df = DataFrame(columns=('A', 'B'))

test002\_df.A = test002\_df.A.astype(int)

test002\_df.B = test002\_df.B.astype(int)

test002\_df.loc[0] = [1,2]

print test002\_df.dtypes

print test002\_df

Output:

A int64

B int64

dtype: object

A B

1. 1 2

## Create a Pivot Table from a DataFrame

data\_df.iloc[:5, :5]



pivoted = data\_df.pivot\_table('Total', index=data\_df.index.time, columns=data\_df.index.date)

pivoted.iloc[:5, :5]



If you plot the pivoted DataFrame, each column plots as a different line:

%matplotlib inline

import matplotlib.pyplot as plt

plt.style.use('seaborn')

pivoted.plot(legend=False, alpha=0.01)



## Delete a Column

df = df.drop('column\_name', 1) # 1 deletes columns

## Delete Rows Having Nulls in Certain Columns

from pandas import DataFrame

import pandas as pd

df = DataFrame(columns=('A', 'B', 'C'))

df.loc[0] = [1, 2, None]

df.loc[1] = [2, 4, 6]

print df, "\n"

missing = df.dropna(subset = ['C'])

print missing

A B C

0 1 2 None

1 2 4 6

A B C

1 2 4 6

## Delete Duplicate Rows

ship\_df = ship\_df.drop\_duplicates()

Note: subset argument allows you to define the columns which define “duplicate”

## Display DataFrame column types

print shipment\_df.dtypes

Output:

mins\_to\_planned\_arr float64

planned\_arrival\_utc datetime64[ns]

pred\_arr\_mins int64

sap\_shipment\_id int64

status\_dt\_utc datetime64[ns]

diff timedelta64[ns]

## Extract a column from a DataFrame into a Series

my\_series = frame.al # al is one of the column names

my\_series

0 en-US,en;q=0.8

1 NaN

2 en-US

3 pt-br

4 en-US,en;q=0.8

5 en-US,en;q=0.8

6 pl-PL,pl;q=0.8,en-US;q=0.6,en;q=0.4

7 bg,en-us;q=0.7,en;q=0.3

## Get the rows in a DataFrame having a null in some column

mins\_rem\_missing\_df = input\_df[pd.isnull(input\_df['MINS\_REMAINING'])]

yields

status\_dt\_utc MINS\_REMAINING KM\_REMAINING HOUR\_OF\_DAY\_UTC \

489 2015-07-08 12:13:51 NaN 326.425442 12

490 2015-07-08 12:28:22 NaN 324.009080 12

491 2015-07-08 12:30:18 NaN 323.348925 12

492 2015-07-08 12:45:13 NaN 323.348202 12

## Fast update of a DataFrame column

from pandas import DataFrame, read\_csv

from cStringIO import StringIO

def append\_something(input):

return(input + "!")

df = read\_csv("test\_df\_data.txt", header=0)

print df

column\_series = df['header3']

modified\_df = column\_series.apply(append\_something)

df['header3'] = modified\_df

print df

>>>

header1 header2 header3

0 a b c

1 d e f

header1 header2 header3

0 a b c!

1 d e f!

>>>

## Filter out na values from a column

frame.al.dropna() # al is a column name in frame

## Find the Row Index in a DataFrame with a Particular Value

import pandas as pd

import numpy as np

df = pd.DataFrame(np.arange(1,7).reshape(2,3),

columns = list('abc'),

index=pd.Series([2,5], name='b'))

print(df)

# a b c

# b

# 2 1 2 3

# 5 4 5 6

print(np.where(df.index==5)[0])

# [1]

print(np.where(df['c']==6)[0])

# [1]

### Row Index Matching Values in Multiple Columns

row\_index\_array = np.where(np.logical\_or(shipment\_df.status\_code == 'X3',

shipment\_df.status\_code == 'AF'))[0]

## Get DataFrame column names

list(df)

-or-

>>> from pandas import DataFrame

>>> journey\_data.columns.values

array(['OWNER', 'VEHICLE\_NUMBER', 'CORRIDOR\_NAME', 'JOURNEY\_NUMBER',

'JOURNEY\_DURATION\_HRS', 'ARRIVAL\_TIME'], dtype=object)

## Get DataFrame column values

from pandas import DataFrame, read\_csv, Series

df = DataFrame(columns=('A', 'B', 'C'))

df.loc[0] = [1,2,3]

df.loc[1] = [1,2,4]

df.loc[2] = [1,7,8]

print df.B.values

print type(df.B)

[ 2. 2. 7.]

<class 'pandas.core.series.Series'>

>>>

## Get dimensions of a DataFrame

>>> type(journey\_data)

<class 'pandas.core.frame.DataFrame'>

>>> journey\_data.shape

(14410, 6)

## Get row count from a DataFrame

>>> len(journey\_data)

14410

## Get rows from a DataFrame by index

>>> import pandas as pd

>>> df[2:4]

OWNER VEHICLE\_NUMBER CORRIDOR\_NAME \

2 Ragos KBH255J BUSIA-KAMPALA

3 COUNTRYMOTORS KBW990K BUSIA-KAMPALA

JOURNEY\_NUMBER JOURNEY\_DURATION\_HRS \

2 SGS-160868-KSM-02 5

3 SGS-KRA-2014NKUC11588-KRA-2014NKUC11588-01 14

ARRIVAL\_TIME

2 11-OCT-13 12.57.19.000000000 PM

3 26-MAR-14 03.08.02.000000000 AM

>>> type(df[2:4])

<class 'pandas.core.frame.DataFrame'>

## Get Rows from a DataFrame which Match an Element of a List

In [5]: df = DataFrame({'A' : [5,6,3,4], 'B' : [1,2,3, 5]})

In [6]: df

Out[6]:

A B

0 5 1

1 6 2

2 3 3

3 4 5

In [7]: df[df['A'].isin([3, 6])]

Out[7]:

A B

1 6 2

2 3 3

## Get unique values from a DataFrame column

from pandas import DataFrame, read\_csv, Series

df = DataFrame(columns=('A', 'B', 'C'))

df.loc[0] = [1,2,3]

df.loc[1] = [1,2,4]

df.loc[2] = [1,7,8]

unique\_val\_list = list(set(df.B))

print unique\_val\_list

>>> corr\_names = DataFrame(journey\_data, columns=['CORRIDOR\_NAME'])

>>> corr\_names

CORRIDOR\_NAME

0 ACCRA SGS OFFICE-ADENTA-MR TANKIA

1 ACCRA SGS OFFICE-PIG FARM JUNCTION TOTAL-ACCRA

2 BUSIA-KAMPALA

…

[14410 rows x 1 columns]

>>> from pandas import Series

>>> Series(corr\_names.values.ravel()).unique()

array(['ACCRA SGS OFFICE-ADENTA-MR TANKIA',

'ACCRA SGS OFFICE-PIG FARM JUNCTION TOTAL-ACCRA', 'BUSIA-KAMPALA',

'BUSIA-MOMBASA', 'CHASE TEMA-SAKAMAN TOTAL-ACCRA', …

## GroupBy Functionality

<http://chrisalbon.com/python/pandas_apply_operations_to_groups.html>

groupby\_carrier = corr\_subset\_df['abs\_err\_mins'].groupby(corr\_subset\_df['carrier'])

print groupby\_carrier.median()

Output:

carrier

DART 116.810555

SCNN 48.376467

USXI 89.106971

Also, mean, std, min, max, count, describe()

counts = df['sap\_shipment\_id'].groupby(df['sap\_shipment\_id']).count()

print type(counts)

print counts.values

print "mean = %f" % np.mean(counts.values)

print "stdev = %f" % np.std(counts.values)

output:

<class 'pandas.core.series.Series'>

[ 1 1 1 ..., 16 9 11]

mean = 16.386067

stdev = 7.048713

### Extract the values from a GroupBy

groupBy.max().values

groupby\_shipment = corr\_df['actual\_stationary\_sec\_rem'].groupby(corr\_df['sap\_shipment\_id'])

print type(groupby\_shipment)

test\_series = groupby\_shipment.max()

print test\_series

print type(test\_series.values)

print test.values[0:10]

Output:

<class 'pandas.core.groupby.SeriesGroupBy'>

sap\_shipment\_id

305983891 53778.058568

305983892 46943.959868

305983893 54163.514338

...

306192132 25738.955543

306192133 59705.278771

306194785 41280.000000

Name: actual\_stationary\_sec\_rem, Length: 100, dtype: float64

<type 'numpy.ndarray'>

[ 53778.05856769 46943.95986826 54163.51433815 0.

124016.33373177 90999.37652502 75965.70095359 0.

51818.71226756 43008.00886809]

### Extract only the rows containing the max value of a column, in a GroupBy

idx = shipment\_lat\_lon\_pdf.groupby(['shipmentId'])['contig\_sta\_hrs'].transform(max) == shipment\_lat\_lon\_pdf['contig\_sta\_hrs']

sta\_shipment\_max\_pdf = shipment\_lat\_lon\_pdf[idx]

sta\_shipment\_max\_pdf = sta\_shipment\_max\_pdf[sta\_shipment\_max\_pdf['contig\_sta\_hrs'] > 0.0]

sta\_shipment\_max\_pdf

Output:



## Insert a column into a DataFrame

owner\_codes = get\_owner\_codes(veh\_subset\_df)

output\_df.insert(1, 'OWNER\_CODE', owner\_codes, allow\_duplicates=True)

## Keep Only Certain Columns of a DataFrame

## Select the ones you want

df1 = df[['a','d']]

## Max value of a DataFrame column

from pandas import DataFrame, read\_csv, Series

df = DataFrame(columns=('A', 'B', 'C'))

df.loc[0] = [1,2,3]

df.loc[1] = [1,2,4]

df.loc[2] = [1,7,8]

print df

print max(df['B'])

A B C

0 1 2 3

1 1 2 4

2 1 7 8

7.0

## Plot the Data in a DataFrame

import pandas as pd

data\_df = pd.read\_csv('Fremont.csv', index\_col='Date', parse\_dates=True)

data\_df.head()

| Fremont Bridge West Sidewalk | Fremont Bridge East Sidewalk |
| --- | --- |
| Date |  |  |
| 2012-10-03 00:00:00 | 4.0 | 9.0 |
| 2012-10-03 01:00:00 | 4.0 | 6.0 |
| 2012-10-03 02:00:00 | 1.0 | 1.0 |
| 2012-10-03 03:00:00 | 2.0 | 3.0 |
| 2012-10-03 04:00:00 | 6.0 |  |

%matplotlib inline

data\_df.plot()

<matplotlib.axes.\_subplots.AxesSubplot at 0x11a812c10>



### Scatter Plot

plt.scatter(sorted\_df['leg\_km\_to\_port\_boundary'], sorted\_df['mins\_to\_leg\_end'] )

plt.xlabel('leg\_km\_to\_port\_boundary')

plt.xlim(12000, 0) **# Reverses the x axis**

plt.ylabel('mins\_to\_leg\_end')

plt.title('km\_to\_port\_boundary vs time')

plt.show()

Output:



## Randomly Split a DataFrame

import numpy as np

import pandas as pd

df = pd.DataFrame(np.random.randn(10, 2))

msk = np.random.rand(len(df)) < 0.8

train = df[msk]

test = df[~msk]

print df

print msk

print type(train)

print train

0 1

0 0.171749 0.556087

1 0.744720 0.923194

2 0.209449 -0.539187

3 0.231416 1.674134

4 -1.426739 -0.492131

5 -0.919245 -0.203287

6 -0.656182 -1.516310

7 1.053216 -0.445366

8 1.641810 0.066485

9 0.449852 -2.599164

[ True False True True True False True True True False]

<class 'pandas.core.frame.DataFrame'>

0 1

0 0.171749 0.556087

2 0.209449 -0.539187

3 0.231416 1.674134

4 -1.426739 -0.492131

6 -0.656182 -1.516310

7 1.053216 -0.445366

8 1.641810 0.066485

## Random Sample from a DataFrame

undersampled\_safe\_loans\_df = safe\_loans\_df.sample(n=bad\_loan\_count, random\_state=0)

## Read a CSV file into a DataFrame

<http://pandas.pydata.org/pandas-docs/stable/generated/pandas.read_csv.html>

from pandas import read\_csv

df = read\_csv('eta\_exp\_2/eta\_corr\_owner\_veh\_kph\_km-rem\_mins-rem\_pace\_lat\_lon\_hr\_day\_min.csv', header=0)

// note header=0 means the column names are read from the first row of the input file

## Re-Sample a DataFrame to Aggregate

print data\_df.head()

weekly\_df = data\_df.resample('W').sum()

print weekly\_df.head()

Fremont Bridge West Sidewalk \

Date

2012-10-03 00:00:00 4.0

2012-10-03 01:00:00 4.0

2012-10-03 02:00:00 1.0

2012-10-03 03:00:00 2.0

2012-10-03 04:00:00 6.0

Fremont Bridge East Sidewalk

Date

2012-10-03 00:00:00 9.0

2012-10-03 01:00:00 6.0

2012-10-03 02:00:00 1.0

2012-10-03 03:00:00 3.0

2012-10-03 04:00:00 1.0

Fremont Bridge West Sidewalk Fremont Bridge East Sidewalk

Date

2012-10-07 7297.0 6995.0

2012-10-14 8679.0 8116.0

2012-10-21 7946.0 7563.0

2012-10-28 6901.0 6536.0

2012-11-04 6408.0 5786.0

## Parse formatted dates while reading a CSV

DATE\_FORMAT\_STRING\_MB = r'%d-%b-%y %I.%M.%S.%f000 %p'

test\_output\_df = read\_csv('mb11\_jrny\_chckpnt\_data.csv',

header=0,

parse\_dates=['FEED\_DT',

'LEG\_PLANNED\_ARR\_DT',

'LEG\_ACTUAL\_ARR\_DT'],

date\_parser = lambda x: pd.to\_datetime(x,

format=DATE\_FORMAT\_STRING\_MB,

coerce=True)

## Remove Rows which Match Elements of a List

df = df[~df['sap\_shipment\_id'].isin(shipments\_to\_remove)]

## Reset DataFrame Data Type

all\_output\_df[['sap\_shipment\_id']] = all\_output\_df[['sap\_shipment\_id']].astype(str)

## Reset DataFrame Indices

model\_input\_df = model\_input\_df.reset\_index(drop=True)

Output:

trimmed\_model\_input\_df:

leg\_km\_to\_port\_boundary timestamp leg\_end\_ts

0 41.591225 1513540761 1513553591

1 1775.726532 1487234207 1487478385

2 1416.337185 1489286955 1489541110

3 1807.828208 1483983265 1484258816

4 3056.373241

## Sample a DataFrame

import pandas

import random

df = pandas.DataFrame(np.random.randn(100, 4), columns=list('ABCD'))

rows = random.sample(df.index, 10)

df\_10 = df.ix[rows]

df\_90 = df.drop(rows)

## Scale Multiple Columns of a DataFrame

X\_df = trimmed\_model\_input\_df[['leg\_km\_to\_port\_boundary', 'sog']]

Output:

X\_df=

leg\_km\_to\_port\_boundary sog

0 778.248402 0.0

1 778.248402 0.0

2 776.868580 14.1

3 775.199350 14.1

...

scaler = StandardScaler()

X\_scaled\_array = scaler.fit\_transform(X\_df) # Returns a 2-d array

print "X\_scaled\_array="

Output:

X\_scaled\_array=

[[ 1.95613761 -1.10259899]

[ 1.95613761 -1.10259899]

[ 1.95113754 0.08856891]

...,

[-0.86400695 -1.043463 ]

[-0.86400695 -1.10259899]

[-0.86400695 -1.08570299]]

## Select a cell from a DataFrame

>>> journey\_data[1:2]

OWNER VEHICLE\_NUMBER CORRIDOR\_NAME \

1 Total Ghana ITDEMO ACCRA SGS OFFICE-PIG FARM JUNCTION TOTAL-ACCRA

JOURNEY\_NUMBER JOURNEY\_DURATION\_HRS \

1 SGS-ITDEMO-20140722-01 2

ARRIVAL\_TIME

1 22-JUL-14 03.47.05.000000000 PM

>>> journey\_data.iloc[1][0]

'Total Ghana'

>>> journey\_data.iloc[1]['CORRIDOR\_NAME']

'ACCRA SGS OFFICE-PIG FARM JUNCTION TOTAL-ACCRA'

## Select rows from a DataFrame by value of a column

>>> journey\_data.shape

(14410, 6)

>>> subset = journey\_data[journey\_data['CORRIDOR\_NAME'] == 'MOMBASA-NAIROBI']

>>> subset.shape

(4481, 6)

## Select rows from a DataFrame by values of multiple columns

from pandas import DataFrame

df = DataFrame(columns=('A', 'B', 'C'))

df.loc[0] = [1,2,3]

df.loc[1] = [1,2,4]

df.loc[2] = [1,7,8]

print df

A B C

0 1 2 3

1 1 2 4

2 1 7 8

df2 = df[(df['A'] == 1) & (df['B'] == 2)]

print df2

A B C

0 1 2 3

1 1 2 4

## Select rows having NaN or null in Multiple Columns

import pandas as pd

import numpy as np

df = pd.DataFrame(columns=('A', 'B', 'C'))

df.loc[0] = range(3)

df.loc[1] = [0, np.NaN, 0]

df.loc[2] = [0, 0, np.NaN]

df.loc[3] = range(3)

print df

df2 = df[(df['B'].isnull() | df['C'].isnull() )]

print '\n', df2

Output:

>>>

A B C

0 0 1 2

1 0 NaN 0

2 0 0 NaN

3 0 1 2

A B C

1 0 NaN 0

2 0 0 NaN

## Sort a DataFrame

**>>>** df = pd.DataFrame({

**...**  'col1' : ['A', 'A', 'B', np.nan, 'D', 'C'],

**...**  'col2' : [2, 1, 9, 8, 7, 4],

**...**  'col3': [0, 1, 9, 4, 2, 3],

**...** })

**>>>** df

col1 col2 col3

0 A 2 0

1 A 1 1

2 B 9 9

3 NaN 8 4

4 D 7 2

5 C 4 3

### Sort by a column

**>>>** df.sort\_values(by=['col1'])

col1 col2 col3

0 A 2 0

1 A 1 1

2 B 9 9

5 C 4 3

4 D 7 2

3 NaN 8 4

### Sort by multiple columns

**>>>** df.sort\_values(by=['col1', 'col2'])

col1 col2 col3

1 A 1 1

0 A 2 0

2 B 9 9

5 C 4 3

4 D 7 2

3 NaN 8 4

### Sort Descending

od\_pair\_pdf = od\_pair\_pdf.sort\_values(by=['count'], ascending=[False])

print od\_pair\_pdf

Output:

od\_pair count

162 (22591, 14662) 7

39 (14662, 22838) 6

33 (14662, 22591) 3

126 (22649, 14662) 3

31 (22744, 22876) 3

## Split a DataFrame into train and test sets

import pandas as pd

import numpy as np

from sklearn.cross\_validation import train\_test\_split

train, test = train\_test\_split(df, test\_size = 0.2)

## Split a DataFrame into train, validate, and test sets

def train\_validate\_test\_split(df, train\_percent=.6, validate\_percent=.2, seed=None):

np.random.seed(seed)

perm = np.random.permutation(df.index) # like "[7 8 2 3 1 6 0 5 4 9]"

m = len(df.index)

train\_end = int(train\_percent \* m)

validate\_end = int(validate\_percent \* m) + train\_end

train = df.iloc[perm[:train\_end]]

validate = df.iloc[perm[train\_end:validate\_end]]

test = df.iloc[perm[validate\_end:]]

return train, validate, test

# end train\_validate\_test\_split()

# test train\_validate\_test\_split()

np.random.seed([42])

df = pd.DataFrame(np.random.rand(10, 5), columns=list('ABCDE'))

train, validate, test = train\_validate\_test\_split(df, seed=42)

assert round(train.iloc[0]['A'], 4) == 0.7046

assert round(validate.iloc[1]['E'], 4) == 0.9572

assert round(test.iloc[1]['B'], 4) == 0.7297

## Substitute for na values in a column

from pandas import DataFrame, Series

import pandas as pd; import numpy as np

frame = DataFrame(records)

frame['tz'].value\_counts()

clean\_tz = frame['tz'].fillna('Missing')

clean\_tz

8

9

10 America/Los\_Angeles

11 America/New\_York

12 America/New\_York

13 Missing

## Summary statistics for a DataFrame

import pandas as pd

df = pd.read\_csv('winequality-red.csv', sep=';', header=0)

df.describe()



## Write a DataFrame to a csv file

veh\_data\_subset.to\_csv(out\_file\_name, header=True, index=False) # index=False suppresses row\_id

### Wrapping CSV file columns in quotes

import csv

output\_df.to\_csv('lat\_lon\_epoch\_weather4.csv', index=False, **quoting=csv.QUOTE\_ALL** )

# Date Functions

## Add a time interval to a datetime

Definition: relativedelta.relativedelta(self, dt1=None, dt2=None,

years=0, months=0, days=0, leapdays=0, weeks=0, hours=0, minutes=0,

seconds=0, microseconds=0, year=None, month=None, day=None,

weekday=None, yearday=None, nlyearday=None, hour=None, minute=None,

second=None, microsecond=None)

>>> from dateutil.relativedelta import relativedelta

>>> import datetime

>>> today = datetime.datetime.today()

>>> today

datetime.datetime(2014, 11, 20)

>>> one\_day\_relative = relativedelta(days=1)

>>> today + one\_day\_relative

datetime.date(2014, 11, 21)

# Add minutes

>>> from dateutil.relativedelta import relativedelta

>>> jetzt = datetime.now()

>>> jetzt

datetime.datetime(2014, 12, 14, 6, 24, 12, 475000)

>>> twenty\_mins\_relative = relativedelta(minutes=20)

>>> twenty\_mins\_relative

relativedelta(minutes=+20)

>>> jetzt + twenty\_mins\_relative

datetime.datetime(2014, 12, 14, 6, 44, 12, 475000)

## Calculate a time interval

>>> from datetime import date, timedelta

>>> date.today()

datetime.date(2014, 8, 25)

>>> thirty\_days\_ago = date.today() - timedelta(days=30)

>>> thirty\_days\_ago

datetime.date(2014, 7, 26)

## Calculate a time interval in seconds, days

>>> t1 = datetime.now()

>>> t1

datetime.datetime(2014, 10, 22, 14, 44, 43, 868000)

>>> t2 = datetime.now()

>>> t2 - t1

datetime.timedelta(0, 20, 740000)

>>> from datetime import datetime

>>> datetime.now()

datetime.datetime(2014, 10, 22, 14, 48, 20, 878000)

t1 = datetime(2015, 9, 9, 0, 0, 0, 0)

t2 = datetime(2015, 9, 9, 0, 2, 0, 0)

assert minutes\_between(t1, t2) == 2

>>> (t2-t1).total\_seconds()

20.74

>>> type(t2-t1)

<type 'datetime.timedelta'>

>>> from datetime import timedelta

>>> test = datetime.now()

>>> test2 = datetime.now()

>>> test

datetime.datetime(2014, 11, 20, 6, 0, 42, 884000)

>>> test2

datetime.datetime(2014, 11, 20, 6, 0, 57, 274000)

>>> d = test2 - test

>>> d

datetime.timedelta(0, 14, 390000)

>>> d.seconds

14

>>> d.days

0

## Convert a datetime to Epoch Seconds

int((datetime.datetime(2012,04,01,0,0) - datetime.datetime(1970,1,1)).total\_seconds())

1333238400

## Convert an Epoch to a time

>>> import time

>>> time.gmtime(1423371600)

time.struct\_time(tm\_year=2015, tm\_mon=2, tm\_mday=8, tm\_hour=5, tm\_min=0, tm\_sec=0, tm\_wday=6, tm\_yday=39, tm\_isdst=0)

## Convert string to date

import datetime

>>> datetime.datetime.strptime("02/05/2014", "%m/%d/%Y").date()

datetime.date(2014, 2, 5)

>>> datetime.datetime.strptime("2/5/2014", "%m/%d/%Y").date()

datetime.date(2014, 2, 5)

>>> datetime.strptime('26-MAR-14 03.08.02.000000000 AM', '%d-%b-%y %I.%M.%S.000000000 %p')

datetime.datetime(2014, 3, 26, 3, 8, 2)

from datetime import datetime

status\_dt = datetime.strptime(journey\_subset\_df.iloc[i]['FEED\_DT'],

'%d-%b-%y %I.%M.%S.000000000 %p')

### Microseconds

import datetime

DATE\_FORMAT\_STRING\_MB = r'%d-%b-%y %I.%M.%S.%f000 %p'

print datetime.strptime("19-AUG-15 09.52.14.167842000 AM", DATE\_FORMAT\_STRING\_MB)

2015-08-19 09:52:14.167842

## Date Time Format Strings

| **Directive** | **Meaning** | **Notes** |
| --- | --- | --- |
| %a | Locale’s abbreviated weekday name. |  |
| %A | Locale’s full weekday name. |  |
| %b | Locale’s abbreviated month name. |  |
| %B | Locale’s full month name. |  |
| %c | Locale’s appropriate date and time representation. |  |
| %d | Day of the month as a decimal number [01,31]. |  |
| %H | Hour (24-hour clock) as a decimal number [00,23]. |  |
| %I | Hour (12-hour clock) as a decimal number [01,12]. |  |
| %j | Day of the year as a decimal number [001,366]. |  |
| %m | Month as a decimal number [01,12]. |  |
| %M | Minute as a decimal number [00,59]. |  |
| %p | Locale’s equivalent of either AM or PM. | (1) |
| %S | Second as a decimal number [00,61]. | (2) |
| %U | Week number of the year (Sunday as the first day of the week) as a decimal number [00,53]. All days in a new year preceding the first Sunday are considered to be in week 0. | (3) |
| %w | Weekday as a decimal number [0(Sunday),6]. |  |
| %W | Week number of the year (Monday as the first day of the week) as a decimal number [00,53]. All days in a new year preceding the first Monday are considered to be in week 0. | (3) |
| %x | Locale’s appropriate date representation. |  |
| %X | Locale’s appropriate time representation. |  |
| %y | Year without century as a decimal number [00,99]. |  |
| %Y | Year with century as a decimal number. |  |
| %Z | Time zone name (no characters if no time zone exists). |  |
| %% | A literal '%' character. |  |

Notes:

1. When used with the [**strptime()**](http://docs.python.org/2/library/time.html#time.strptime) function, the %p directive only affects the output hour field if the %I directive is used to parse the hour.
2. The range really is 0 to 61; this accounts for leap seconds and the (very rare) double leap seconds.
3. When used with the [**strptime()**](http://docs.python.org/2/library/time.html#time.strptime) function, %U and %W are only used in calculations when the day of the week and the year are specified.
4. %f indicates microseconds.

### Another method:

>>> from dateutil.parser import \*

>>> from dateutil.tz import \*

>>> parse('2014-09-11 13:11:07+00:00')

datetime.datetime(2014, 9, 11, 13, 11, 7, tzinfo=tzutc())

## Create an arbitrary datetime

datetime.datetime(2014, 10, 22, 14, 48, 20, 878000) # yr, mo, day, hr, min, sec, us

### datetime with time zone

import pytz

dt\_utc = datetime(2015, 10, 21, 4, 40, 0, 0, pytz.UTC)

print dt\_utc

2015-10-21 04:40:00+00:00

## Get the current datetime

>>> from datetime import datetime

>>> datetime.now()

## Get year, month, day, hour, minute, second, milliseconds, weekday

>>> from datetime import datetime

>>> t2

datetime.datetime(2014, 10, 22, 14, 45, 4, 608000)

>>> t2.year

2014

>>> t2.month

10

>>> t2.day

22

>>> t2.hour

14

>>> t2.minute

45

>>> t2.second

4

>>> t2.microsecond

608000

>>> today = datetime.now()

>>> today

datetime.datetime(2014, 10, 23, 9, 30, 37, 743000)

>>> today.weekday()

3

[Note: In python, Monday is 0, Sunday is 6]

## ISO Weekday

ISO weekday is Monday = 1, Sunday = 7

>>> from datetime import datetime

>>> jetzt = datetime.now()

>>> jetzt

datetime.datetime(2015, 3, 2, 9, 33, 14, 907000)

>>> jetzt.isoweekday()

1

## Time Zone Names

<https://en.wikipedia.org/wiki/List_of_tz_database_time_zones>

# Dictionaries

Used as has tables or associative arrays

## Convert a DataFrame to a Dictionary

>>> test = DataFrame(columns=('A', 'B'))

>>> test.loc[0] = ['Turlock', 'CA']

>>> test.loc[1] = ['Lexington', 'KY']

>>> test

A B

0 Turlock CA

1 Lexington KY

>>> test\_dict = test.set\_index('A').to\_dict()

>>> test\_dict

{'B': {'Turlock': 'CA', 'Lexington': 'KY'}}

>>> test\_dict['B']['Turlock']

'CA'

## Create a dictionary

>>> test = {}

>>> test[(1,2)] = 1

>>> (1,2) in test

True

## Execute a Function on all the Values in a Dictionary

d2 = {k: f(v) for k, v in d1.items()}

## Get a value for a key in the dict

dict = {'Name': 'Zara', 'Age': 27}

print "Value : %s" % dict.get('Age')

print "Value : %s" % dict.get('Sex', "Never")

Value : 27

Value : Never

## Get the keys from a dictionary

>>> my\_dict = {}

>>> my\_dict['A'] = 1

>>> my\_dict

{'A': 1}

>>> my\_dict.keys() # note: returns a list

['A']

## Is a key in a dictionary?

>>> test = {}

>>> test["something"] = 1

>>> test["something"]

1

>>> "somethingelse" in test

False

>>> "something" in test

True

>>> test["somethingelse"]

Traceback (most recent call last):

File "<pyshell#5>", line 1, in <module>

test["somethingelse"]

KeyError: 'somethingelse'

# Directories

## Check if a Directory exists

import os

print(os.path.isdir("/home/el"))

## Concatenate a Directory and File Name

import os

base\_dir = r'c:\bla\bing'

filename = r'data.txt'

os.path.join(base\_dir, filename)

'c:\\bla\\bing\\data.txt'

## Create a Directory

if not os.path.exists('kept\_models\_by\_corr'):

os.makedirs('kept\_models\_by\_corr')

## Delete all the files and folders in a directory

import os

import shutil

def clear\_dir(folder):

""" Delete all the file sin the specified path name.

Path name will be like '/path/to/folder'

"""

for root, dirs, files in os.walk(folder):

for f in files:

os.unlink(os.path.join(root, f))

for d in dirs:

shutil.rmtree(os.path.join(root, d))

## Delete all the files in a directory

import os

def clear\_dir(folder):

""" Delete all the file sin the specified path name.

Path name will be like '/path/to/folder'

"""

for the\_file in os.listdir(folder):

file\_path = os.path.join(folder, the\_file)

try:

if os.path.isfile(file\_path):

os.unlink(file\_path)

except Exception, e:

print e

## Get the Current Working Directory

>>> import os

>>> os.getcwd()

'C:\\Python27'

## Read the files in a directory.

path=r"C:\Users\bbeauchamp\Documents\Data Analytics\Customers\_and\_Projects\SGS\eta\_raw\parser"

>>> import os

>>> files = os.listdir(path)

File names are returned as elements in a list. Note that this will also read in subdirectories.

### Read the files in a directory with a specific extension

>>> import glob

>>> my\_list = glob.glob(r"C:\Users\bbeauchamp\Documents\Data Analytics\Customers\_and\_Projects\SGS\eta\_raw\parser\\*.arff") # note returns a list

## Set the working directory

>>> import os

>>> os.chdir('c:/dev/python')

>>> os.getcwd()

'c:\\dev\\python'

# Exception Handling

## try-except

**import** **sys**

**try**:

f = open('myfile.txt')

s = f.readline()

i = int(s.strip())

**except** IOError **as** e:

**print** "I/O error({0}): {1}".format(e.errno, e.strerror)

**except** ValueError:

**print** "Could not convert data to an integer."

**except**:

**print** "Unexpected error:", sys.exc\_info()[0]

**raise**

## Print the traceback and stack trace

import traceback

try:

# Create the scaler

scaler = preprocessing.StandardScaler().fit(X\_for\_scaler)

# Store the scaler serialization for use during prediction

ser\_string = pickle.dumps(scaler, pickle.HIGHEST\_PROTOCOL)

text\_file = open(scaler\_output\_dir + corr\_name + ".txt", "wb")

text\_file.write(ser\_string)

text\_file.close()

except:

print "Error - Unable to generate scaler:", sys.exc\_info()[0]

print "filepath=%s" % filepath

print "X\_for\_scaler:"

for j in xrange(0, len(X\_for\_scaler)):

print "row[%d] = %s" % (j, str(X\_for\_scaler[j]))

print "\*\*\* print\_exc:"

traceback.print\_exc()

raise

# Files

## Copy a file between from one directory to another

from shutil import copyfile

copyfile(filename, new\_file\_name)

## Copy a File from a URL

URL = 'https://data.seattle.gov/api/views/65db-xm6k/rows.csv?accessType=DOWNLOAD'

import urllib

urllib.urlretrieve(URL, 'Fremont.csv')

!head 'Fremont.csv'

Output:

Date,Fremont Bridge West Sidewalk,Fremont Bridge East Sidewalk

10/03/2012 12:00:00 AM,4,9

10/03/2012 01:00:00 AM,4,6

## Delete a file

os.remove("corr\_validation.csv")

## Does a file exist?

import os.path

os.path.isfile(fname)

## Extract the Filename and Extension from a path

import os

filename, file\_extension = os.path.splitext('/path/to/somefile.ext')

>>> filename

'/path/to/somefile'

>>> file\_extension

'.ext'

## Extract the file name from a path

>>> pathname = "C:\Users\bbeauchamp\Documents\Data Analytics\Customers\_and\_Projects\SGS\eta\_raw\parser\F000008\_km\_min.arff"

>>> from os.path import basename

>>> print basename(pathname)

F000008\_km\_min.arff

or

import ntpath

ntpath.basename("training\_data\_by\_corr\@GENERIC\_31.37\_-92.41\_39.90\_-84.26\_train.csv")

'@GENERIC\_31.37\_-92.41\_39.90\_-84.26\_train.csv'

## Open File dialog

import Tkinter, tkFileDialog

root = Tkinter.Tk()

root.withdraw()

file\_path = tkFileDialog.askopenfilename()

## Read a text file into a string

with open('model\_train.sql', 'r') as model\_train\_sql\_file:

model\_train\_sql = model\_train\_sql\_file.read()

## Read all the lines in a file into a list

>>> text\_file = open("eta\_corr\_owner\_veh\_km-rem\_mins-rem\_pace\_lat\_lon\_hr\_day\_mo\_no-dupe.csv", "r")

>>> lines = text\_file.readlines()

>>> print len(lines)

614161

>>> print lines[1]

BUSIA-MOMBASA,PURA LOGISTICS,SLMKBN434B,37,397.0563678,1402,7.189250437,-1.49094,37.0571,14,6,9

## Read a text file line by line

filename = 'calamp\_msg\_out\_3.xml'

with open(filename) as f:

for line in f:

print(line)

## Read a CSV file

import csv

with open('C:\\Users\\bbeauchamp\\Documents\\Data Analytics\\Customers\_and\_Projects\\SGS\\' +

'GSM\_fail\_2\_or\_more\_devices\_with\_conn\_pct.csv', 'rb') as csvfile:

myReader = csv.reader(csvfile, delimiter = ',')

print( 'test')

for myRow in myReader:

print', '.join(myRow)

***Unzip Files***

import zipfile

fname\_list = os.listdir(DATA\_DIR)

# Remove hidden operating system files

ipaws\_fname\_list = filter(lambda fname: fname[0] != '.', fname\_list)

# Create the output directory if it doesn't already exist

if not os.path.exists(UNZIPPED\_DIR):

os.makedirs(UNZIPPED\_DIR)

# end if

# Now go through, unzip each zipped file, and store it to the unzipped directory

for zipfile\_name in ipaws\_fname\_list:

path = os.path.join(DATA\_DIR, zipfile\_name)

zf = zipfile.ZipFile(path)

# The zipfile may have multiple files in it, so read through them and extract each one.

# Here we really only expect one raw file per zip file

for info in zf.infolist():

# Read the data in from the file

data = zf.read(info.filename)

# And write it out to the output file

output\_file\_name = os.path.join(UNZIPPED\_DIR, basename(info.filename))

uncompressed\_file = open(output\_file\_name, 'w')

uncompressed\_file.write(data)

uncompressed\_file.close()

# next info

# next zipfile

## Write to a Text File

points\_file = open('C:\\Users\\bbeauchamp\\Documents\\Data Analytics\\Customers\_and\_Projects\\SGS\\test\_output', 'w')

points\_file.write('This is a test\n')

points\_file.close()

# Geocoding

import urllib

import json

def get\_lat\_lon(address):

print("starting")

params = { }

# params[ 'key' ] = "AIzaSyAfHtyiQmO7OpAp8WiM8RzGcBlYQqCo67w" # the actual key, of course, is not provided here

params[ 'sensor' ] = "false"

params[ 'address' ] = address

params = urllib.urlencode( params )

print "http://maps.googleapis.com/maps/api/geocode/json?%s" % params

f = urllib.urlopen( "http://maps.googleapis.com/maps/api/geocode/json?%s" % params )

reply = f.read()

decodeddata = json.loads(reply)

latitude = (decodeddata['results'][0]['geometry']['location']['lat'])

longitude = (decodeddata['results'][0]['geometry']['location']['lng'])

return([latitude, longitude])

print(get\_lat\_lon("1439 Buckeye Court Auburn CA"))

# Geography

## Distance between two coordinates

import geopy

from geopy.geocoders import Nominatim

from geopy.distance import vincenty

point\_a = (42.52574, -71.42404)

point\_b = (42.526, -71.42644)

print vincenty(point\_a, point\_b).km

# Hash Functions

>>> import hashlib

>>> print hashlib.sha1("This is a test").hexdigest()

a54d88e06612d820bc3be72877c74f257b561b19 # this is a string object

# Images

## View an Image using matplotlib

import numpy as np

import matplotlib.pyplot as plt

from PIL import Image

fname = 'image.png'

image = Image.open(fname).convert("L")

arr = np.asarray(image)

plt.imshow(arr, cmap='gray')

plt.show()

# Installing packages

## easy\_install

c:\Python27\Scripts>easy\_install googlemaps

Searching for googlemaps

Reading https://pypi.python.org/simple/googlemaps/

Reading http://sourceforge.net/projects/py-googlemaps/

…

zip\_safe flag not set; analyzing archive contents...

Adding googlemaps 2.0 to easy-install.pth file

Installed c:\python27\lib\site-packages\googlemaps-2.0-py2.7.egg

Processing dependencies for googlemaps

Finished processing dependencies for googlemaps

**Note: You must then close IDLE and reopen it.**

# json

## Pretty Print JSON

import JSON

print json.dumps(data, indent=4)

outputs:

{

"accountTotal": 122124.19,

"receivedLateFees": 18.61,

"receivedInterest": 71092.26,

"infundingBalance": 900,

"outstandingPrincipal": 121141.38,

"investorId": 918209,

"receivedPrincipal": 176669.6,

"accruedInterest": 1940.7,

"availableCash": 82.81,

"totalPortfolios": 2,

"totalNotes": 8942

}

## Reading a json file into a dict

import json

fname = r"c:/temp/test6.json"

my\_json\_file = open(fname)

my\_data = json.load(my\_json\_file)

print(my\_data["algorithmData"]["conveyance"]["id"])

# Jupyter Notebooks

## Display an Image inside a Notebook

from IPython.display import Image

image\_fname = 'notMNIST\_large/A/a2ltaWRvcmkgbXVnY3VwLnR0Zg==.png'

Image(filename=image\_fname)

## Display matplotlib plots inline in the notebook

%matplotlib inline

## Store a CSV file in the local directory (not the HDFS directory)

from pandas import DataFrame

test\_df = shipment\_modes\_df.toPandas() # shipment\_modes\_df is a pyspark DataFrame

test\_df.to\_csv('test\_csv.csv', index=False)

# Lambdas

## Conditional Lambdas

products['contains\_perfect'] = products['perfect'].apply(lambda x: 1 if x >0 else 0)

# Libraries

## Find the Function Available in a Library

import math

dir(math)

# Lists

## Apply Functions to the Elements of a List

# Convert predicted stationary seconds remaining to sta minutes remaining

rsm\_predictions = [pred\_seconds / 60.0 for pred\_seconds in rs\_sec\_predictions]

## Average of items in a list

l = [15, 18, 2, 36, 12, 78, 5, 6, 9]

sum(l) / float(len(l))

mean = np.average(np.array(tot\_stationary\_mins\_list, dtype=float))

## Cartesian Product of 2 Lists

from itertools import product

list1 = [1,2,3]

list2 = [4,5,6]

list(product(list1, list2))

Output:

[(1, 4), (1, 5), (1, 6), (2, 4), (2, 5), (2, 6), (3, 4), (3, 5), (3, 6)]

## Concatenate 2 lists

>>> a = [1,2]

>>> b = [3,4]

>>> a+b

[1, 2, 3, 4]

## Concatenate the string elements of 2 Lists

list1 = ['a', 'b']

list2 = ['1', '2']

map(lambda(x,y): x+y, zip(list1, list2))

Output:

['a1', 'b2']

## Copy a list

import copy

new\_list = copy.deepcopy(old\_list)

## Create a list containing a number of constants

>>> [1] \* 10

[1, 1, 1, 1, 1, 1, 1, 1, 1, 1]

>>>

## Convert a list to a dict

dict([('A', 1), ('B', 2), ('C', 3)])

{'A': 1, 'C': 3, 'B': 2}

## Count the Number of Occurences of an Item in a List

test = ['A', 'B', 'C', 'A']

print test.count('A')

>>>

2

## Creating and Appending to a List

>>> adds = []

>>> adds

[]

>>> adds.append("a")

>>> adds

['a']

>>> adds.append("b")

>>> adds

['a', 'b']

>>> 'a' in adds

True

>>> 'c' in adds

False

## Filter a List

non\_stop\_words = filter(lambda word: word not in stopwords, tokens)

Note: The conditional in the lambda indicates the elements you want to **keep** in the list.

## Last items in a List

a = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]

a

output:

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]

a[-9:]

output:

[4, 5, 6, 7, 8, 9, 10, 11, 12]

## Merge 2 Lists with Option to Remove Dupes

def merge\_lists(list1, list2, remove\_dupes=False):

''' Combine the lists, keeping only unique items if desired

'''

output\_list = list1 + list2

if remove\_dupes:

output\_list = list(set(output\_list))

# end if

return(output\_list)

# end merge\_lists()

# test merge\_lists()

assert merge\_lists([], [], 'True') == []

assert merge\_lists([], [], 'False') == []

assert merge\_lists(['A'], ['A', 'B']) == ['A', 'A', 'B']

assert merge\_lists(['A'], ['A', 'B'], True) == ['A', 'B']

assert merge\_lists(['A'], ['A', 'B'], False) == ['A', 'A', 'B']

## Randomly Split a List

import numpy as np

from sklearn.cross\_validation import train\_test\_split

x = range(11)

print x, type(x)

x\_train, x\_test = train\_test\_split(x, test\_size=0.3, random\_state=0)

print x\_train

print x\_test

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10] <type 'list'>

[6, 1, 7, 8, 3, 0, 5]

[4, 9, 2, 10]

## Randomly Sample Items from a List

import random

random.sample(the\_list, 50)

## Randomly Sample Items from a List with Replacement

import random

np.random.choice(3,10)

Output:

array([1, 0, 0, 0, 1, 1, 0, 2, 1, 2])

## Remove an Item from a List

aList = [123, 'xyz', 'zara', 'abc', 'xyz'];

aList.remove('xyz');

print "List : ", aList

aList.remove('abc');

print "List : ", aList

Output:

List : [123, 'zara', 'abc', 'xyz']

List : [123, 'zara', 'xyz']

## Remove Null Values from a List

Use a list comprehension

>>> a = [1,2,3,None]

>>> b = [x for x in a if x is not None]

>>> b

[1, 2, 3]

## Replace an item in a list

import numpy as np

list1 = [1,2,3,None,5]

print list1

for i,val in enumerate(list1):

if val is None:

list1[i] = 17

print list1

>>>

[1, 2, 3, None, 5]

[1, 2, 3, 17, 5]

>>>

## Sort a list

>>> b = [1,4,2,7,3,8]

>>> b

[1, 4, 2, 7, 3, 8]

>>> b.sort()

>>> b

[1, 2, 3, 4, 7, 8]

>>> sorted([5, 2, 3, 1, 4])

[1, 2, 3, 4, 5]

## Shuffle the items in a list

>>> test = [1,2,3,4,5]

>>> from random import shuffle

>>> shuffle(test)

>>> test

[2, 3, 1, 4, 5]

## Subtract the Elements in 2 Lists

import numpy as np

corr\_subset\_df['pred\_error\_mins'] = list(np.array(pred\_mins\_to\_arr\_list) -

np.array(mins\_to\_arr\_actual))

## Standard Deviation of items in a list

import numpy as np

sd = np.std(np.array(tot\_stationary\_mins\_list, dtype=float))

## Using lambda and map on a list

squares = map(lambda x: x\*\*2, range(10))

print squares

output:

[0, 1, 4, 9, 16, 25, 36, 49, 64, 81

# Machine Learning

## Create Word Count columns

# Split out the words into individual columns

for word in important\_words:

products[word] = products['review\_clean'].apply(lambda s : s.split().count(word))

## Euclidean Distance

from sklearn.metrics.pairwise import euclidean\_distances

counts = [

[0, 1, 1, 0, 0, 1, 0, 1],

[0, 1, 1, 1, 1, 0, 0, 0],

[1, 0, 0, 0, 0, 0, 1, 0]

]

print 'Distance between 1st and 2nd documents:', euclidean\_distances(counts[0], counts[1])

print 'Distance between 1st and 3rd documents:', euclidean\_distances(counts[0], counts[2])

print 'Distance between 2nd and 3rd documents:', euclidean\_distances(counts[1], counts[2])

Distance between 1st and 2nd documents: [[ 2.]]

Distance between 1st and 3rd documents: [[ 2.44948974]]

Distance between 2nd and 3rd documents: [[ 2.44948974]]

## One-Hot Encoder

from sklearn.feature\_extraction import DictVectorizer

onehot\_encoder = DictVectorizer()

instances = [

{'city': 'New York'},

{'city': 'San Francisco'},

{'city': 'Chapel Hill'}

]

print onehot\_encoder.fit\_transform(instances).toarray()

# Maps

## folium to easily create a map

import folium

map\_bruce = folium.Map(location=[38.8966, -121.0769], zoom\_start=13) # center of map, my town

folium.Marker([38.8752, -121.0763], popup='House').add\_to(map\_bruce) # Location of my house

folium.Marker([38.8953, -121.0785], popup='Closest Pub').add\_to(map\_bruce) # Location of the closest pub

map\_bruce



# Math Functions

## Exponentiation

>>> 2\*\*3

8

## Largest float

>>> sys.float\_info.max

1.7976931348623157e+308

## Median

>>> import numpy as np

>>> a = np.array([1,2,3,4])

>>> np.median(a)

2.5

## Modulo

>>> 17 % 3

2

## pi

math.pi

## Random Numbers

### Random float

>>> import random

>>> random.seed(0)

>>> random.uniform(0, 0.2)

0.10956309498898086

### Set the Random Number Seed

## import random

random.seed(42)

random.seed(None) # Seeds from current time

## Rounding

### General rounding

>>> round(110.574388557174, 3)

110.574

### Round to half-even

import decimal

>>> decimal.Decimal('2.675').quantize(decimal.Decimal('.01'), rounding=decimal.ROUND\_HALF\_EVEN)

Decimal('2.68')

>>> decimal.Decimal('2.665').quantize(decimal.Decimal('.01'), rounding=decimal.ROUND\_HALF\_EVEN)

Decimal('2.66')

Note: **To round to a float equivalent of an integer**, use ‘0’ for the decimal.Decimal() argument.

def round\_half\_even(floating\_point\_str, how\_many\_dec\_points):

""" Rounds the specified floating point value (encoded as a string)

to the specified number of decimal points, using

ROUND\_HALF\_EVEN rounding method.

2.675 rounds to 2.68

2.665 rounds to 2.66

Requires import decimal

"""

rounding\_arg = decimal.Decimal(str(10 \*\* (-1 \* how\_many\_dec\_points)))

rounded = (decimal.Decimal(floating\_point\_str)

.quantize(rounding\_arg,

rounding=decimal.ROUND\_HALF\_EVEN))

return(rounded)

#

OR

def round\_half\_even(some\_float):

''' Performs even half rounding of floating point values '''

some\_float\_str = str(some\_float)

half\_rounded\_even = decimal.Decimal(some\_float\_str).quantize(decimal.Decimal('0.01'),

rounding = decimal.ROUND\_HALF\_EVEN)

return(float(half\_rounded\_even))

# end round\_half\_even()

def self\_test():

""" Tests the code in this python program

"""

# test round\_half\_even()

assert round\_half\_even('2.675', 2) == Decimal('2.68')

assert round\_half\_even('2.665', 2) == Decimal('2.66')

print "self test complete"

### Round to {x.0, x.5} intervals

def round\_to\_half(float\_arg):

""" Round a floating point number to the nearest 1/2,

e.g round\_to\_half(1.25) = 1.0 but

round\_to\_half(1.255) = 1.5

"""

twice = float\_arg \* 2.0

rounded = decimal.Decimal(twice).quantize(decimal.Decimal('0'), rounding=decimal.ROUND\_HALF\_EVEN)

return(float(rounded) / 2.0)

## Scale a 2-Dimensional List

from sklearn.preprocessing import StandardScaler

data = [[0, 1],

[1, 2],

[2, 4],

[3, 6]]

print "type(data)= ", type(data)

print "data=: ",data

scaler = StandardScaler()

print "scaler.fit(data):"

print(scaler.fit(data))

print

print("mean: ", scaler.mean\_)

print("scale: ", scaler.scale\_)

print

print "scalar.transform(data):"

print(scaler.transform(data))

print

print("scaler.transform([[0.5, 7]]): "), scaler.transform([[0.5, 7]])

Output:

type(data)= <type 'list'>

data=: [[0, 1], [1, 2], [2, 4], [3, 6]]

scaler.fit(data):

StandardScaler(copy=True, with\_mean=True, with\_std=True)

('mean: ', array([ 1.5 , 3.25]))

('scale: ', array([ 1.11803399, 1.92028644]))

scalar.transform(data):

[[-1.34164079 -1.1717002 ]

[-0.4472136 -0.65094455]

[ 0.4472136 0.39056673]

[ 1.34164079 1.43207802]]

scaler.transform([[0.5, 7]]): [[-0.89442719 1.95283366]]

## Square Root

import math

math.sqrt(25)

## Test for nan

from math import isnan

nan\_float = float(‘nan’)

>>> math.isnan(nan\_float)

True

# Matrices

## Number of rows in a matrix

row\_count = X\_train.shape[0]

## Read a Matrix from a file

>>> import numpy as np

>>> my\_data = np.genfromtxt(filepath, delimiter=',', skip\_header=8)

(my\_data will be a 2d numpy array)

## Read the contents of a matrix column into an array

>>> mydata

array([[ 1. , 19.91142191, 16. , 74. ],

[ 2. , 17.99404762, 15. , 48. ],

[ 3. , 18.94845361, 16. , 89. ],

[ 4. , 29.55978261, 21.5 , 120. ],

[ 5. , 25.80927835, 18. , 129. ],

[ 6. , 20.21631206, 16. , 16. ],

[ 7. , 18.47900763, 15. , 15. ],

[ 8. , 18.82753165, 15. , 30. ],

[ 9. , 16.14227642, 15. , 21. ],

[ 10. , 18.10933941, 15. , 37. ],

[ 11. , 18.24694377, 14. , 151. ],

[ 12. , 17.70260223, 12. , 79. ]])

>>> rainfall\_mm = mydata[:,3]

>>> rainfall\_mm

array([ 74., 48., 89., 120., 129., 16., 15., 30., 21.,

37., 151., 79.])

## Scale matrix columns

from sklearn import preprocessing

scaler = preprocessing.StandardScaler().fit(X)

X\_scaled = scaler.transform(X)

# Methods

## Method Header Template

def brokenTen(value):

"""Incorrect implementation of the ten function.

Note:

The `if` statement checks an undefined variable `val` instead of `value`.

Args:

value (int): A number.

Returns:

bool: Whether `value` is less than ten.

Raises:

NameError: The function references `val`, which is not available in the local or global

namespace, so a `NameError` is raised.

"""

# numpy

## Covariance

xbar = (6 + 8 + 10 + 14 + 18) / 5

ybar = (7 + 9 + 13 + 17.5 + 18) / 5

cov = ((6 - xbar) \* (7 - ybar) + (8 - xbar) \* (9 - ybar) + (10 - xbar) \* (13 - ybar) +

(14 - xbar) \* (17.5 - ybar) + (18 - xbar) \* (18 - ybar)) / 4

print cov

import numpy as np

print np.cov([6, 8, 10, 14, 18], [7, 9, 13, 17.5, 18])

print np.cov([6, 8, 10, 14, 18], [7, 9, 13, 17.5, 18])[0][1]

22.65

[[ 23.2 22.65]

[ 22.65 24.3 ]]

22.65

## Element-Wise Multiplication of two Arrays or Iterables

import numpy as np

weighted\_err = np.multiply(uc1\_df.legacy\_med\_abs\_err, uc1\_df.legacy\_shipment\_count)

## r-squared

from sklearn.linear\_model import LinearRegression

X = [[6], [8], [10], [14], [18]]

y = [[7], [9], [13], [17.5], [18]]

X\_test = [[8], [9], [11], [16], [12]]

y\_test = [[11], [8.5], [15], [18], [11]]

model = LinearRegression()

model.fit(X, y)

print 'R-squared: %.4f' % model.score(X\_test, y\_test)

R-squared: 0.6620

## Variance

import numpy as np

X = [6,8,10,14,18]

print np.var(X, ddof=1)

23.2

# Object Serialization

## Create an object from a stored serialization

# Load the dictionary back from the pickle file.

import pickle

favorite\_color = pickle.load( open( "latlong.p", "rb" ) )

print(favorite\_color["330 Lee Industrial Blvd Austell, Ga 30168"])

print(favorite\_color["330 LEE INDUSTRIAL BLVD AUSTELL, GA 30168"])

## Store and then Recall a Binary Object

Note that for some objects (GridSearchCV objects, for example), pickle.dump works, but pickle.dumps fails.

test921 = [[1.0, 2.0, 3.0, 4.0],

[2.0, 3.0, 4.0, 5.0]]

test922 = [[1.0, 2.0, 3.0, 4.0],

[2.0, 4.0, 6.0, 8.0]]

# Create a new scaler, and train on test921

my\_scaler = preprocessing.StandardScaler().fit(test921)

print "my\_fitted\_data on test922 ="

results\_922 = my\_scaler.transform(test922)

print "original\_scaler=", results\_922

pickle.dump(my\_scaler, open('pickle\_test\_921.p', 'wb'))

# Now load the scaler from the serialized file, and repeat the test on test922

recalled\_scaler = pickle.load(open('pickle\_test\_921.p', 'rb'))

results\_486 = recalled\_scaler.transform(test922)

print "recalled\_scaler = ", results\_486

print "recalled data identical?: ", results\_486 == results\_922

## Serialize and Store an Object

tzw\_str = pickle.dumps(tzw, pickle.HIGHEST\_PROTOCOL)

text\_file = open("tzwhere\_tz\_converter.txt", "wb")

text\_file.write(tzw\_str)

text\_file.close()

## Simple Example - Save and Recall a dictionary

<https://wiki.python.org/moin/UsingPickle>

# Save a dictionary into a pickle file.

import pickle

favorite\_color = { "lion": "yellow", "kitty": "red" }

pickle.dump( favorite\_color, open( "save.p", "wb" ) )

# Load the dictionary back from the pickle file.

favorite\_color = pickle.load( open( "save.p", "rb" ) )

# favorite\_color is now { "lion": "yellow", "kitty": "red" }

## Store a Binary Object

pickle.dump( favorite\_color, open( "save.p", "wb" ) )

# Packages

## Check Package Version

>>> import statlib

>>> print statlib.\_\_version\_\_

# pandas

## Change the number of rows printed for pandas objects

import pandas as pd

def print\_full(x):

pd.set\_option('display.max\_rows', len(x))

print(x)

pd.reset\_option('display.max\_rows')

## pandasql

Uses SQLite3 in the background, so syntax permitted by SQLite3 should work

### Installing pandasql

c:\>pip install -U pandasql

Collecting pandasql

Downloading pandasql-0.6.3.tar.gz

Installing collected packages: pandasql

Running setup.py install for pandasql

Successfully installed pandasql-0.6.3

### Querying using pandasql

import pandas as pd

from pandasql import sqldf

from pandasql import load\_births # returns a test dataset

births\_df = load\_births()

q = """

SELECT \*

FROM births\_df b

where b.date > '1976-01-01'

and b.date < '1977-01-01'

"""

births\_1976\_df = sqldf(q, globals())

print births\_1976\_df

>>>

date births

0 1976-01-01 00:00:00 259173

1 1976-01-01 00:00:00 257455

2 1976-02-01 00:00:00 238153

3 1976-02-01 00:00:00 236551

…

21 1976-11-01 00:00:00 258011

22 1976-12-01 00:00:00 265787

23 1976-12-01 00:00:00 265886

>>>

## Set the max number of columns displayed when showing a DataFrame

import pandas as pd

pd.set\_option('display.max\_columns', 50)

Later...

print transit\_file\_df.head(20) # shows all the columns in the DataFrame

# Plotting

import matplotlib.pyplot as plt

## Histograms

savi\_error = get\_error\_list(df, 'PRED\_MINS\_REM', 'MINS\_REMAINING')

plan\_error = get\_error\_list(df, 'PLAN\_ETA\_MINS\_REM', 'MINS\_REMAINING')

my\_bins = range(-3000, 3000, 100)

plt.hist(savi\_error, histtype='step', color='b', label='Savi',

normed=False, bins=my\_bins)

plt.hist(plan\_error, histtype='step', color='r', label='Plan',

normed=False, bins=my\_bins)

plt.title("ETA Error: Savi ETA vs Planned Arrival Time")

plt.xlabel("Error, minutes")

plt.ylabel("Frequency")

plt.legend()

plt.show()



## Line + Scatter Plots using plotly

from plotly.offline import download\_plotlyjs, init\_notebook\_mode, plot, iplot

from plotly.graph\_objs import Scatter, Figure, Layout

import plotly.plotly as py

import plotly.graph\_objs as go

init\_notebook\_mode(connected=True)

def plot\_battery\_level(df):

"""Plots a polyline of distance traversed from Origin over Time from ShipmentStart"""

trace = go.Scatter(

x = df['hdr\_ts'],

y = df['battery'],

mode = 'lines+markers'

)

data = [trace]

layout = go.Layout(

yaxis=dict(title='Battery Level (%)'),

xaxis=dict(title='Date/Time (UTC)'),

margin=go.Margin(t=10)

)

fig = go.Figure(data=data, layout=layout)

iplot(fig)

print "Plotting battery level over time..."

plot\_battery\_level(raw\_reads)



## Scatter plot

import pandas as pd

import matplotlib.pylab as plt

df = pd.read\_csv('winequality-red.csv', sep=';')

plt.scatter(df['alcohol'], df['quality'])

plt.xlabel('Alcohol')

plt.ylabel('Quality')

plt.title('Alcohol vs Quality')

plt.show()



# Program Execution

## Stopping program execution

from sys import exit

exit('exiting....')

# pyspark

## pyspark.sql.dataframe.DataFrame

### Column Names

od\_pair.columns

Output:

['org', 'source\_id', 'dest\_id', 'start\_date', 'tk\_owner', 'lane\_enabled', 'end\_datep']

### Extract a List from a pyspark.sql.dataframe.DataFrame Column

unique\_vessel\_id\_list = list(set(vessels\_df.select("vesselId").rdd.flatMap(lambda x: x).collect()))

print len(unique\_vessel\_id\_list)

print unique\_vessel\_id\_list

Output:

84

[u'9243162', u'9757876', u'9695145', u'9335202', u'9419644', u'9776183', u'9395147', u'9695169', u'9336191', u'9112686', ...]

### Read in a file from the server’s local OS (not HDFS)

from pandas import DataFrame

import pandas as pd

pg\_lanes\_df = pd.read\_csv('pg\_lanes\_export\_2017-05-18.csv', header=0)

### Show the values in an array column

test\_df = sqlContext.sql(

''' SELECT distinct modes

FROM shipment\_sums\_table

WHERE legCount > 1

LIMIT 5

''')

test\_df.select('modes').collect()

Output:

[Row(modes=[u'truck', u'truck']),

Row(modes=[u'truck', u'truck', u'truck', u'truck']),

Row(modes=[u'truck', u'truck', u'truck']),

Row(modes=[u'intermodal', u'intermodal'])]

## pyspark version

sc.version

## Write a pySpark DataFrame to a CSV File

legacy\_perf\_df.toPandas().to\_csv('CSCL\_ATLANTIC\_OCEAN.csv', index=False, quoting=csv.QUOTE\_ALL)

# Python version

import platform

print(platform.python\_version())

Output:

2.7.13

# Regular expressions

## Remove punctuation

import re

mystring = "test !123"

out = re.sub('[^A-Za-z0-9 ]+', '', mystring)

# Random Numbers

## Random Integer in a Range

from random import randint

print randint(0,10) # rand int in the range 0,10 inclusive

## Choose Random Items from a List

rs = np.random.choice(unique\_shipment\_ids, size=num\_shipments\_for\_train, replace=False)

## Create a list containing some random numbers

from random import randint

random\_indices = []

for x in range (0, 1000):

random\_indices.append(randint(0, 999))

# REST Services

## Consume a REST service

import json

import urllib2

json.load(urllib2.urlopen("url"))

## Consume an XML Service

import urllib2

EAS\_URL = ("https://apps.fema.gov/" +

"IPAWSOPEN\_EAS\_SERVICE/rest/public/recent/2012-08-21T11:40:43Z?" +

"pin=<suppressed for security>")

# Initial testing of the API

request = urllib2.Request(EAS\_URL)

response = urllib2.urlopen(request)

response\_text = response.read()

print len(response\_text)

# scikit-learn

## GroupKFold Implementation for Cross-Validation

from sklearn import metrics

import numpy as np

from sklearn.model\_selection import GroupKFold

#generate data

X = [0.1, 0.2, 2.2, 2.4, 2.3, 4.55, 5.8, 8.8, 9, 10]

y = ["a", "b", "b", "b", "c", "c", "c", "d", "d", "d"]

groups = [1, 2, 3, 4, 5, 1, 2, 3, 4, 2]

print "groups=", groups

print

gkf = GroupKFold( n\_splits=5)

for train, test in gkf.split(X, y, groups=groups):

print("%s %s" % (train, test))

Output:

groups= [1, 2, 3, 4, 5, 1, 2, 3, 4, 2]

[0 2 3 4 5 7 8] [1 6 9]

[0 1 2 4 5 6 7 9] [3 8]

[0 1 3 4 5 6 8 9] [2 7]

[1 2 3 4 6 7 8 9] [0 5]

[0 1 2 3 5 6 7 8 9] [4]

## Linear regression

from sklearn.linear\_model import LinearRegression

# Training data

X = [[6], [8], [10], [14], [18]]

y = [[7], [9], [13], [17.5], [18]]

# Create and fit the model

model = LinearRegression()

model.fit(X, y)

print 'A 12" pizza should cost: $%.2f' % model.predict([12])[0]

A 12" pizza should cost: $13.68

## sklearn Version

import sklearn

print('The scikit-learn version is {}.'.format(sklearn.\_\_version\_\_))Series (pandas)

## Check if a Series value is null

pd.isnull(df\_row[target\_field\_name])

## Convert a Series to a DataFrame

Series.to\_frame(name=None)

## Create a Series of random numbers

s = Series(np.random.randn(5))

## Get the value of a Series element

>>> this\_journey\_number[0:0]

Series([], name: JOURNEY\_NUMBER, dtype: object)

>>> this\_journey\_number.values

array(['SGS-KRA-2014NKUC11588-KRA-2014NKUC11588-01'], dtype=object)

>>> this\_journey\_number.values[0]

'SGS-KRA-2014NKUC11588-KRA-2014NKUC11588-01'

# SFrame

## Add a Column Based on Other Columns

# Add payment\_inc\_ratio

loans['payment\_inc\_ratio'] = loans.apply(lambda row: row['installment'] \* 12.0 / (row['annual\_inc'] + 0.01))

## Convert an SFrame to features and labels in a numpy array

Two arrays are returned: one representing features and another representing class labels.

Note: The feature matrix includes an additional column 'intercept' filled with 1's to take account of the intercept term.

import numpy as np

def get\_numpy\_data(data\_sframe, features, label):

data\_sframe['intercept'] = 1

features = ['intercept'] + features

features\_sframe = data\_sframe[features]

feature\_matrix = features\_sframe.to\_numpy()

label\_sarray = data\_sframe[label]

label\_array = label\_sarray.to\_numpy()

return(feature\_matrix, label\_array)

feature\_matrix\_train, sentiment\_train = get\_numpy\_data(train\_data, important\_words, 'sentiment')

feature\_matrix\_valid, sentiment\_valid = get\_numpy\_data(validation\_data, important\_words, 'sentiment')

## Copy an Sframe

copy\_sframe = some\_frame.copy()

## First n rows of an Sframe

products.head(10)['name']

## One-Hot Encoding of an Sframe

loans\_data = risky\_loans.append(safe\_loans)

for feature in features:

loans\_data\_one\_hot\_encoded = loans\_data[feature].apply(lambda x: {x: 1})

loans\_data\_unpacked = loans\_data\_one\_hot\_encoded.unpack(column\_name\_prefix=feature)

# Change None's to 0's

for column in loans\_data\_unpacked.column\_names():

loans\_data\_unpacked[column] = loans\_data\_unpacked[column].fillna(0)

loans\_data.remove\_column(feature)

loans\_data.add\_columns(loans\_data\_unpacked)

## Random Split an SFrame

train\_data, validation\_data = products.random\_split(.9, seed=1)

## Remove a Column from an Sframe

Sframe.remove\_column(column\_name)

## Select Rows from an Sframe

example\_labels = sframe.SArray([-1, -1, 1, 1, 1])

subset = example\_labels[example\_labels[:] == -1]

print len(subset)

Response: 2

# Statistics

## Applying lowess smoothing

import numpy as np

import pylab as plt

import statsmodels.api as sm

x = np.linspace(0,2\*np.pi,100)

y = np.sin(x) + np.random.random(100) \* 0.2

lowess = sm.nonparametric.lowess(y, x, frac=0.1)

print type(lowess)

print lowess

plt.plot(x, y, '+')

plt.plot(lowess[:, 0], lowess[:, 1])

plt.show()



## Precision, recall, F1, support

import numpy as np

from sklearn.metrics import precision\_recall\_fscore\_support

#y\_true = np.array([0, 1, 0, 0, 1, 0])

#y\_pred = np.array([0, 1, 1, 0, 0, 1])

y\_true = np.array(['early', 'late', 'early', 'early', 'late', 'early','late'])

y\_pred = np.array(['early', 'late', 'late', 'early', 'early', 'late','late'])

precision, recall, f1, x = precision\_recall\_fscore\_support(y\_true, y\_pred, pos\_label='late', average='micro')

print precision, recall, f1

# Strings

## Concatenate strings

>>> 'is' + 'test'

'istest'

## Convert a character to its ASCII integer

>>> ord('A')

65

## Convert to float

>>> test = "123.45"

>>> type(test)

<type 'str'>

>>> test\_float = float(test)

>>> test\_float

123.45

>>> type(test\_float)

<type 'float'>

## Convert to lower case

>>> "Test".lower()

'test'

## Find a sub-string

>>> test = "something"

>>> test.find('me')

2

## Find nth Occurrence of a sub-string

def find\_nth(haystack, needle, n):

start = haystack.find(needle)

while start >= 0 and n > 1:

start = haystack.find(needle, start+len(needle))

n -= 1

return start

# end find\_nth

assert find\_nth('abcdef\_def\_def', 'def', 0) == 3 # nonsense, just return first index

assert find\_nth('abcdef\_def\_def', 'def', 1) == 3

assert find\_nth('abcdef\_def\_def', 'xyz', 1) == -1

assert find\_nth('abcdef\_def\_def', 'def', 4) == -1

## Formatted strings

>>> test = "something %s" % "great"

>>> test

'something great'

### Right-Pad a Numeric Formatted String with zeros

x = 1.500000

print '%.2f' % x

print '{:.3f}'.format(x)

Output:

1.50

1.500

## Remove Punctuation

>>> test = "some sunava #%^@$#"

>>> import string

>>> test.translate(None, string.punctuation) # may not work on Spark

'some sunava '

-or-

import re

output\_str = re.sub(r'[^\sa-zA-Z0-9]', '', text)

## Replace a substring

>>> test

'out with the old'

>>> test.replace('old', 'new')

'out with the new'

>>>

## String Literals

>>> x = r"\n"

>>> x

'\\n'

>>>

## Sub-strings

>>> x = "Hello World!"

>>> x[2:]

'llo World!'

>>> x[:2]

'He'

>>> x[:-2]

'Hello Worl'

>>> x[-2:]

'd!'

>>> x[2:-2]

'llo Worl'

## Tokenize a string

>>> test = "the world is at my fingertips"

>>> test.split(' ')

['the', 'world', 'is', 'at', 'my', 'fingertips']

## Trim leading and trailing characters

>>> '"Strip the "leading" and "trailing" double quotes"'.strip('"')

'Strip the "leading" and "trailing" double quotes'

>>> test = "A()"

>>> test

'A()'

>>> test.strip("()")

'A'

## Trim white space

>>> test = " something "

>>> len(test)

11

>>> len(test.strip())

9

# Timers

## Sleep

import time

>>> print time.ctime(); time.sleep(10); print time.ctime()

Fri Sep 26 09:09:30 2014

Fri Sep 26 09:09:40 2014

## Timing Code Execution

>>> import time

>>> start\_time = time.time()

>>> print (time.time() - start\_time)

18.8489999771

# Tuples

## Cartesion product of two tuples

import itertools

x = [1, 3, 5]

p = list( itertools.product(x, x))

print str(p)

[(1, 1), (1, 3), (3, 1), (3, 3)]

## Product of the elements in a tuple

from operator import mul

a = (1,2,3,4)

print(reduce(mul, a))

>>>

24

## Verify 2 tuples contain the same elements

print ((22668, 22884) == (22668, 22884))

print ((22884, 22668) == (22668, 22884))

print (sorted((22884, 22668)) == sorted((22668, 22884)))

print type (sorted((22884, 22668))) # returns a list

Output:

True

False

True

<type 'list'>

# User Input

## Get user input from the keyboard

>>> test\_var = raw\_input("Enter some data")

Enter some datacomething

>>> test\_var

'comething'

# XML

import xml.etree.ElementTree

for xml\_file\_name in xml\_fname\_list:

xml\_file\_path = os.path.join(UNZIPPED\_DIR, xml\_file\_name)

root = xml.etree.ElementTree.parse(xml\_file\_path).getroot()

for alert in root:

info = alert.find('{urn:oasis:names:tc:emergency:cap:1.2}info')

if info is not None:

category = info.find('{urn:oasis:names:tc:emergency:cap:1.2}category').text

event = info.find('{urn:oasis:names:tc:emergency:cap:1.2}event').text

Notes:

This XML structure was like

alerts # root

alert

info

category

event

{urn:oasis:names:tc:emergency:cap:1.2} is the namespace

# Errors

|  |  |
| --- | --- |
| ValueError: Cannot set a frame with no defined index and a value that cannot be converted to a Series | Trying to do slicing on an empty DataFrame |
|  |  |
| ValueError: The truth value of a DataFrame is ambiguous. Use a.empty, a.bool(), a.item(), a.any() or a.all(). | Use:  df531['at\_orig'] = df531.apply(get\_at\_orig\_flag, args=(poi\_df\_531,), axis=1)  instead of:  df531['at\_orig'] = df531.apply(get\_at\_orig\_flag, args=(poi\_df\_531), axis=1) |
|  |  |
|  |  |
|  |  |
|  |  |