Cheatography

Data_Analysis Cheat Sheet

by Glowie via cheatography.com/209809/cs/45178/

Academic Integrity

Collaboration - must write own solutions. Group work - acknowledge contributions. Unacknowledged →collusion or plagiarism

Collusion - working together, identical/similar solutions

Plagiarism - using code, ideas, words, data without acknowledgement

Avoid plagiarism - could be unintentional, cite sources - URL, data accessed, code "-adapted from"

Open Source - duplicate license in code

Contract Cheating - using third party

External Resources - used as learning support (stackoverflow). Solution repos (former students work) disallowed

Python

Paradigms - OOP, functional, structured, procedural

Jupyter - Julia, Python, R (.ipynb)

Statement - instruction executed by interpreter i.e. print(), assignment,

Expression - values/variables/operators, represents single result value, can be used on the right side of assignment statements

print() - displays value of expression, not evaluating, no "quotes"

Dynamic Programming - real time execution, without compiling

Dynamically Typed - type not declared, determined at runtime, not compile time

Strongly Typed - checks to ensure type safety, can't typecast anything

Python2

If statement

```
if <ex pr>:
   statements
elif/else:
   statements
```

Function args - keyword (unordered/usually optional, omit for default) or positional

Function definition:

```
def <na me> (params):
   statements
```

Compound statement - header & body

Param - variables declared in functions, Args - values passed to functions

None type - no value : `return None, return, pass`

Traceback/Stack Trace - prints program, line, functions when an error occurs

Type Conversion `float() int() str() eval(< str ing >)`

Python Strings

Array of bytes, no char type - immutable, ordered

Concat - `+`, Repeat - `*`

F-String `f"String {<py code/v ar> }"`

String Format - `" String {one}, {two}".format (one=, two=)`

Indexing - 0 based, []

Slicing - `[start :en d:s tep], [inclu sive, exclusive]` Omitting start/end `[:]`

For loop

```
for c in string:
   statements
```

In operator - membership returns T/F

Methods - `var.me tho dNa me( args)`

Git

Git - Distributed VCS, clients mirror full repo, fault tolerant vs Centralised - single server

Git converts directory to VFC - versioned filesystem, provides operations

Git (cont)

Snapshots, not deltas - changed files are stored, unchanged files use pointers

Only needs local files, can push with network

Git directory, metadata + object db for project

Secure Shell Protocol(SSH) - public/private keys for secure exchange over network

Encrypt with public key, decrypt with private

Data Science

Answer questions/solve problems with data

Hypothesis then collect data, or find dataset, explore, generate hypothesis

Data wrangling/munging - cleaning data, uses pipeline, new data = rerun or extend

EDA - no hypothesis or model, use graphs and summary stats

Can reveal unclean data, more processing

Analysis - model for explanatory research (cause relationships)

Matplotlib

Object orientated - figures and axes

Figure - canvas, set dimensions, background c place objects on it and save

Axes - frontend for plots, placed onto figures

Init - `plt.fi gure()`, `figsize` for size

Add plot `ax1 = fig.ad d_s ubp lot()` u `_su bpl ots (2,2)` for 2x2 grid

`ax1.cl ear()` clear plot

`axes[0 ,0].hist()` Plots to top left, [0,1] = right, [1,0] = bottom left

`.plot(x,y, linest yle=, color=, ma )`

Plot range `ax.set _xl im(x1, x2)`

Matplotlib (cont)

Tick locations `ax.set _xt ick s([])`

Tick labels `ax.set _xt ick lab els([])`

Plot labels `.set_x /yl abel() .set_t itle( ) .legend()`

Saving `fig.sa vef ig( " fil e.png, dpi= )`

Seaborn

Countplot - for categorical, y=freq, x=categories, type of barplot

Scatterplot - 2 cont. variables, shows relationship

Linear reg. scatter + line to model relationship

Barplot - x=categories, y=continuous i.e. means of each category, swap x and y to make horizontal

Can apply hue to count, scatter, bar, box - splits categories using another column

Boxplot - for cont. shows the spread

Can use `.set(t itl e=, xla bel =,y label=)`

UNIX

Written in Assembly, then rewritten in C

UNIX Filesystem - hierarchical, tree structure with nodes - root = /

Nodes - has metadata, at least a name. Leaf Nodes - no children

Path - sequence of nodes to id other nodes in the tree

Absolute Path - sequence of nodes from the root - resolves to a location

Relative Path - starts navigation at current location - intermediate node
Label - traverse to child, .. to parent, . stay

Non-leaf nodes = directory, Leaf nodes = directory/files

UNIX (cont)

Subdirectory - directory inside directory

Files - stores information - name, contents, location, privileges

UNIX Shell - program allows users to interact with UNIX system

Terminal - physical hardware, input + output, dumb terminal/thin client relies on host computer

Terminal emulator - program, text-only on GUI

Terminal needs a SHELL to run commands

SHELL - text only, access OS, executes commands, interacts files, scripting

CLI - style of interface, text-only, runs shell

Check running shell `echo $0`

Data Structures

List - ordered, mutable, dynamic array, heterogenous, head→tail, uses []

Tuple - ordered, immutable, static array, uses ()

Dict - unordered, hashable key-value, associative array, mutable {}

Set - unordered, mutable, unique objects {}

Sort list - `myList.sort()` modifies list, `so rted (my List)` creates new sorted list

List for loop (same as string)

Concat lists with `+`

Dict - uses hash function, converts key to an index to retrieve value - O(1) insertion, access, deletion

Retrieve value by key `myDict ["ke y"]`

Keys must be unique, else they overwrite values

Remove key-value pair `del myDict ["ke y"]`

Dict for loop - `for k,v in myDict.it e ms():`

Data Structures (cont)

Can sort list of k,v pairs using `sorted (di ct.i te ms())`

Sets - add values - `set.ad d(new val)`

Unique values using `set(obj)`

Intersection - `set1.i nte rse cti on( set2)`

Files

Permissions - UID (user id), GID(id for group of users)

Access Rights - Class (who can access) and type (type of access)

Class - u (user) = owner, g (group) = user in group, o (other) = anyone else with access

Type - r (read), w (write), x (execute)

`chmod` change mode - absolute or symbolic mode

Symbolic - class(u, g, o, a), type(r, w, x), op (+, -, =) `chmod g+r *.txt` `chmod u=rw go= file`

Absolute - 3 digits (user, group other), 4=r, 2=w, 1=x `chmod 730 a.txt` u=rwx, g=wx `chmod 641 a.txt` u=rw, g=r, o=x

`cp <sr c1> <sr c2> <ta rge t>`

`cp -r <sr c> <ta rg>` directory + its contents

`cp -r <sr c>/ <ta rg>` contents only

`mv <sr c1> <sr c2> <ta rg>` moves/renames

`rm <pa th1> <pa th2>` by rel/abs path

`rmdir <di r>` removes empty directory

`rm -r <di r>` non-empty directory

Git Anatomy

Tree - snapshot along the timeline in VFS

Commit - snapshot in timeline, has a tree & hash(SHA1) code (id)

Repo - commit in ./git, commit is a tree

Data_Analysis Cheat Sheet

by Glowie via cheatography.com/209809/cs/45178/

Git Anatomy (cont)

Staging area - index, stores info for next commit - chosen changes

Working directory - UNIX directory, files in repo, checked out version

Snapshot - records a treem all files in project at point in tiem

Branch - alt dev path - ptr to latest in its timeline, default = main branch

Init, no parents. Merges, 2. Most have 1

Head - commit we are on, can be ptr to any commit, normally main branch

Git Cycle

3 Stages - modified(changes since checkout), staged(modified and added), committed(in git directory)

Status - untracked, modified, unmodified, staged

Working directory→staging→commit to repo

Repo → working directory (checkout)

Tracked - last snapshot or staged

Untracked - not in last snapshot and not staged

New file (untracked) →staged→committed

Existing file→modifed→committed

Unmod→removed(repo)→untracked

Un/Staged at the same time - modify, stage changes, modify again

Git History/Undo

`git restore <fi le>` discards changes in WD

`git restore --staged <fi le>` unstage, modifications stay, previously tracked are now modified, else untracked

`git rm <fi le>` removes from staging and WD

`git rm --cached <fi le>` removes from stage

Git History/Undo (cont)

`git diff` - WD vs staged `--stage` staged vs last commit

`git log` - latest at top, commits = hashcode

`--pret ty= for mat :""` %h(hash), %an(author) %ar(time) %s(message) `--gr aph`

`git show` metadata, edits, file content of latest or `<co mmi t>` for specific, or HEAD~n

`git checkout HEAD~n <fi le>` copies to WD, all changes lost

Numpy

n-dimensions, homogenous, ordered

Init - list, tuples, np methods

arange - `[inclu sive, exclusive]`

linspace - `[inclu sive, inclusive]`

Vectorized Operations - faster, implemented in C, contiguous memory, parallel processing

Attributes - dtype, ndim, shape, size

Iteration - nested loop or single loop `d.fla t`

Broadcasting - compatible if equal dim or dim of 1 i.e. (4,3) - (), (3,), (1,3), (4,1)

Operations(mean, max..) by axis. Row ↓: `d .mean (ax is=0)` Column→: `axis=1`

Indexing - single el `[row, col]`, entire row `[1,]`, rows `[0:2]`, rows select col `[0: 2,1]`, rows cols `[0:2,1:3]`

Shallow - view, slice, reshape, ravel

Deep - copy, resize, flatten

Pandas Series

1D, homogenous, indexed (default 0 → n-1)

Like fixed length, ordered dicts, maps index to values

Can initialize with dicts - with custom index, autofills with NaN

Can use `in` by index `'a' in d`

`d.index` → indexes, `d.array` → array

Custom index - `pd.Ser ies ([1,2], in dex= ['b ','a'])`

Reindex - to change order/drop indexes `d.r ein dex ([' a', 'b'])`

Indexing - single `d['a']` multi `d[['a' ,'b ']]`

Vectorized operations on values

Index alignment with operations on 2 series with overlapping indexes `d1 + d2`

Update indexes with `d.index = [new in dexes]`, same size only

Forward fill `d.rein dex ([i], method ="ff ill ")`

SHELL cmds

`pwd` - present working directory

`ls <pa th>`, `cd <pa th>`, `mkdir <pa th>`

`mkdir -p foo/ba r/final`

manual `man ls` `man man`

`ls -a` all files (hidden ones)

Wildcards `ls main*` - selectively show files, `main.c main.o main`

`ls <pa th1> <pa th2>` lists files of 2 paths

`ls -l` long format - perms, owner/owner group, size (bytes), modify date

`ls -R` current and all subdirectories

Dataframes Groupby

SAC (split, apply, combine)

Split - by key, done on an axis

Apply - function to each group

Combine results into new object

Keys can be list/array of values (same len as axis) or single value (col)

`df[" dat a1"].gr oup by( df[ " key 1" ]).sum()` returns a series, key1 as the index, sum of data1 as value

`df.dat a1.g ro upb y([ df[ " key 1"],` `df["key2"]]).mean()` returns series with multi index

Unstack multi index with `.unstack()` first index = row, second = column

Group entire df and apply function to all columns `df.gro upb y("k ey1 " ).m ean()`

Error if a column is categorical, can't apply fn

Use `groupby` and `.size` to count values in column (categorical)

`df.gro upb y("k ey") [co l].op` returns series

`df.gro upb y("k ey") [[c ol]].op` returns df

Dataframes np functions

`np.add(df, 10)` adds 10 to all values `df .add()` is the same

`np.abs(df) np.sqr t(df)`

`df.app ly(fn, axis=)` for custom functions, axis=0 by default

`np.sum(), df.sum(), np.mean(), df.m ean()`

Dataframes

2D arrays (higher with hierarchical indexing)

Custom rows and columns indices, can have missing data, column = pd.Series

Dataframes (cont)

Construct with dict equal length lists/arrays

```
dict = {"co l1" : ["va l1", "val2"],
        " col 2" : ["va l1", " val 2"]}
```

Reorder and add columns `pd.Dat aFr ame (data, column s=[ " new "])`

Retrieve col as series `df[" col 1"]`

Update all col values with assignment `df[" col "] = 2` or can use series

Update with series - index matching with df

Can use boolean op to create new col `df[" wes ter n"] = df.pro vince == " Alb ert a"`

Delete column `del df[" wes ter n"]`

`df.rei nde x([])` to rearrange rows/cols, add new ones

`df.drop()` - default rows, `axis=1` for cols

`df.ilo c[r ow,col] [inc, ex],[[1,2] ,[0,2 ]]` for specific indexes

`df.loc ["ro w","c ol"] [inc, inc]`

`grades >= 50` returns df with T/F values

Boolean indexing `grades [(g rades >= 50) &/| (grades < 70)]`

CSV

`with` statement assigns csv file to var, uses it, then closes to release resource

`open()` opens file, and assigns to var

`mode="w "` for writing `r` for reading

`newlin e=""` processes new lines

`csv.wr ite r(file)` and `csv.re ader()`

`writer.wr ite row([])`

`for row in reader: x1, x2.. = row`

`df = pd.rea d_c sv( " fil e", name s=[])` names=columns