**Python cheatsheet**

|  |  |
| --- | --- |
| **Easter egg** | Import this |
| **Import pandas** | import pandas as pd |
| **Import NumPy** | import numpy as np |
| **Series (a column)** | pd.Series([21, 22, 23, 24], name = ‘age’)  *age = column name* |
| **Select an element (row; from 0 to …)** | table\_name[3]  *Select the 4th row* |
| **Make a custom axis to a Series** |  |
|  |  |
| **Index** is also called an **axis**; each element is called axis label.  Data in columns is called as **values**.  A series ideally should have the same datatype throughout its values (same format for the whole column). |  |
| **Dataframe** | A series is a column and a dataframe has multiple columns |
| **Create a dataframe** |  |
| **Columns and rows in a dataframe (table)** | df.shape  (3, 7)  3 rows and 7 columns |
| **Axis 0**  **Axis 1** | Vertical axis (rows)  Horizontal axis (column names) |
| **Select a column** | df[‘ADDRESS’]  Returns a series |
| **Select several columns** | df[[‘ADDRESS’, ‘CITY’]] |
| **Retrieve axis 1 / column names / information** | df.columns  Index([‘first\_name’, ‘last\_name’, ‘email’], dtype=‘object’) |
| **Retrieve axis 0 / vertical axis information** | df.index  RangeIndex(start=0, stop=3, step=1) or min 0, max 3 (3 rows) |
| **Retrieve a row** |  |
| **Make a column as an index** | E.g. you want to filter by last name. Instead of the default 0, 1, 2 etc. you can make the index as Nield, Scala Morrison.  *df.set\_index(‘column\_name’, inplace = True*) inplace is true so it edits the existing df instead of creating a new one  *df.loc[‘Scala’]* you can now use the last name to search for this row  **df.reset\_index(inplace = True)** reset the axis to default |
| **Copy a dataframe** | df2 = df.copy() |

**Importing Data**





**CSV**

Read csv as text:



Read csv into as a pandas dataframe:



**Read csv alternative:**

df = pd.read\_csv('https://raw.githubusercontent.com/thomasnield/machine-learning-demo-data/master/regression/winequality-red.csv')

df

**SQL** [**pd.read\_sql**](https://pandas.pydata.org/docs/reference/api/pandas.read_sql.html)

Import an SQL database:



Alternative:



If you don’t *parse\_dates*:

With parsing and without:



**JSON**

Read a JSON file:



Read a JSON file as a pandas dataframe:



**Selecting Rows and Columns**

**loc and iloc**

* **loc** works on **labels** assigned to the axis, **iloc** works on **numbers**.
* E.g. if axis is surname then **loc** will work only on the surname like ‘**vanli’** while **iloc** will only work on **numbers**.

**Select** (first two) **rows** df.iloc[0:2] or df.iloc[:2]

**Exclude first row** df.iloc[1:]

**Select all** df.iloc[:]

**All rows and columns 2-3** df.iloc[:, 1:3] you are selecting columns 2 and 3 with indexes 1 and 2

**Select last two columns** df.iloc[:, -2:] count from 0 to -1, -2 etc. from right to left. Select the column you want to have and with : you will select everything to the right

**Select the last row** df.iloc[-1]

**Select all rows and last column** df.iloc[:, -1]

**Select all rows and all columns except for the last column** df.iloc[:, :-1]

If you want to get the first 2 rows you need to select the third index, in this case 2:



**Select specific columns and rows**. In this case you will select column index 0 and column index 2 (first\_name and email). In the second query you will select second row and third column



**Select rows and columns using loc** df.loc[["samiam","thomasnield"], "email"]

**Reset index** df.reset\_index(inplace = True)

**Select values that start with a specific letter** condition = df["username"].str.startswith("s") username = column, s = letter start

df[condition]

**Multiple conditions. AND & OR are & and |** condition = df["username"].str.startswith("s") & df["email"].str.contains("gmail")

df[condition]

**Drop columns and rows**





**Remove columns:**



**Remove columns by selecting specific columns like column 1 and 4:**



**Adding rows and columns (appending)**

**Add a column at the end of the dataframe:**



**Add a column at a specific place:**



**Add a row:**



**Using concat (merge two datasets):**



**Updating data**

**Making a column in caps lock (upper):**



**Updating on a condition:**



**Update row on condition:**



**Unpivoting data (melting):**



Id\_vars = untouched columns

Value\_vars = columns that will be unpivoted

Var\_name = column names will be moved to this column

Value\_name = column values will be moved to this column



**Sorting, Casting, and Categories**

**Datatypes:**



**Timestamp example (date)** pd.Timestamp(‘20230130’) → 2023-01-30

**Difference between days** pd.Timestamp('20230130') - pd.Timestamp('20230127') → 3 days

**View datatypes** df.dtypes



You will get column name on the left (float, int, datatime etc.) and datatype on the right.

**Change a column to a different datatype** df[‘columnname’] = df[‘columnaname’].astype(‘bool’)

**Sort by 2 columns (first lightning, then rain inches)** df.sort\_values(by=["lightning","rain\_inches"])

df.sort\_values(by=["lightning","rain\_inches"],ascending=[False,True])

df.sort\_values(axis=0, ascending = True) *ascending if one value*

When using the sort methods, remember to **add the inplace=True parameter if you want to replace the existing dataframe** with the sorted one.

Sort by an row index or columns index df.sort\_index(axis = 0) for rows

df.sort\_index(axis = 1) for columns

**Replace a column with a *category* datatype:**

cat\_type = pd.CategoricalDtype(categories=["CLEAR", "MINOR", "MAJOR", "SEVERE"], ordered=True)

df["severity"] = df["severity"].astype(cat\_type) severity = column you want to replace

df.sort\_values(by=["severity"])

If you apply a categorization on a column that has values not mapping to any category, then those will become NA values.

**Python if/elif/else category function:**



**Apply this function to the wind\_speed\_mph column:**



**Add a new column:**



Categorize the last column and sort the data by that column in DESC:



**Removing Duplicative and Sparse Data**

**df used:**



**Get duplicated rows using .duplicated() function:**  df.duplicated()



It will mark rows that are duplicates.

If you want to see original rows and their duplicates (like in Excel) then add (keep=False):



**Look for duplicates in a specific column using *subset*:**



Or if you want to use multiple columns:



**Delete duplicates** df.drop\_duplicates(inplace=True)

**Delete duplicates based on a column** df.drop\_duplicates(subset=['record\_id'], inplace=True)

Number of unique values in a column df.nunique()



**Identify columns with single-values (e.g. value Shop in the whole column):**



**Drop these columns:** df.drop(delete\_cols, axis=1, inplace=True)

**Read csv file (open csv file) (using a link):**

wine\_df = pd.read\_csv('https://raw.githubusercontent.com/thomasnield/machine-learning-demo-data/master/regression/winequality-red.csv')

wine\_df

**Get number of rows and columns from .shape function:**



**Alternatively. Get number of rows ([0]) and columns ([1]) from the .shape function (where X = df):**



**Count the number of unique values per column:**



**Remove columns with 5% or less unique values:**



**Alternative using scikit-learn, VarianceThreshold and fit\_transform():**



Return from ndarray to get the columns using get\_support:



**Remove columns with duplicates and 3 or less unique values:**



**f-string / f’’:**



**Handling missing data**

**Looking for missing values**

**Looking for missing values per row** df.isna() *not efficient in my opinion, alternatives:* df.notna(), df.isnull(), df.notnull()



**Check columns for missing values** df.isna().any() *or you can specify axis*  any(axis=0)



**Check rows for missing values** df.isna().any(axis = 1)



**Select (show) columns with NaN (null) values** df.loc[:, df.isna().any()]



**Select (show) rows with NaN (null) values** df.loc[df.isna().any(axis=1), :] *or* df.loc[df.isna().any(axis=1)]



**Look for missing values in specific columns** df[df['TEMPERATURE'].isna() | df['RAIN'].isna()]



**Removing rows with missing values**

Note: many ML and statistical models do not tolerate NA, NaN or other missing null values.

**Remove rows with missing values / drop rows with missing values** df.dropna(axis=0, inplace=True)

**Remove rows with missing values in specific column** df.dropna(axis=0, subset=["RAIN"], inplace=True)

**Remove rows with missing values in specific columns** df.dropna(axis=0, subset=[“TEMPERATURE”, "RAIN"], inplace=True)

**Remove columns with missing values / drop columns with missing values** df.dropna(axis=1, inplace=True)

**Replacing missing values**

**Replace missing values with -1** df.fillna(value=-1, inplace=True)

You can’t specify a column using the subset parameter. To target specific columns you will need to extract them out and then apply fillna().

**NUMPY Replace values with NaN** from numpy import nandf.replace(-1, nan, inplace=True)



**SCIKIT LEARN Replace missing values with mean using SimpleImputer (imputer.fit, imputer.transform)**





There are other options for the *strategy* parameter including 'mean', 'median', 'most\_frequent', and 'constant'.