Pandas cheat sheet

Good to know

In Python, everything is an object. We read our data into a *dataframe object*. Each column of this dataframe object is a *Series object*. Some functions of pandas applies to dataframe objects and some apply to Series objects.

Similarly some return dataframe objects and some return Series objects. When a function returns a Series object you can assign this to a new column on your dataframe to create a new column. E.g.:

```
df['new_col'] =
df['column'].function_that_returns_
series_object()
```

Getting information on columns

```
df['column'].value_counts()
```

# Returns the counts of how many times each unique value occurs in this column

```
df['column'].mean()
```

# Returns the mean of all the values of the column. Only applies to numerical columns. Alternatives are min(), max(), std() and median.

```
df.sort_values('column_name')
```

# Returns the whole dataframe where all the rows are sorted based on the values of given column.

# Default is ascending sort but can be changed by adding "ascending=False" as a parameter.

# If you give a list of column names, it will sort the dataframe with all the columns in order.

Reading the data

```
df = pd.read_csv('name_of_file.csv')
```

# Reads the file from your local file system and writes it into a variable called df.

Getting information on the dataframe

```
df.head()
```

# Returns the first five rows of the dataframe. It is possible to specify number of columns to be returned as a parameter inside the brackets.

```
df.shape
```

# Does not need parentheses. Returns a tuple with number of rows and columns respectively.

```
df.describe()
```

# Returns common statistics such as mean, standard deviation etc on every column.

```
df.columns
```

# Returns the name of all the columns in a list

```
df.index
```

# Returns the index of the dataframe

```
df.dtypes
```

# Returns the types of each column. Common types are: integer, float, string (listed as Object), datetime.

```
df.isna().sum()
```

# Shows how many missing values there are on each column.

Don't forget to refer to the original documentation for more parameters and details on these functions.

Changing values of the dataframe

```
pd.to_datetime(df['column_name'])
```

# Returns a Series object where the values in the given column are casted into datetime objects.

# This way, it become easier to extract date, year, month, day, hour and minute information from these values. Example:

```
df['datetime_column'].dt.year
```

# This will return a Series object with year values of these timestamps. You can assign it to a new column on your dataframe.

```
df['column_name'].astype(int)
```

# Returns a Series object with the column type changed into integer. You can also cast columns to float or string with this function.

```
df['column_name'].apply(lambda x: True if x == 5 else False)
```

# Returns a Series object. Each value in the given column is run through the function and values are returned accordingly. In this example, the result will be a Series object where values will be True or False depending on if the value in the column is greater or smaller than 5.

```
df['column_name'].replace('70 – 80%.', 0.7)
```

# The occurrence of the first given value is replaced with the second given value. Returns a Series object.

```
df['column_name'].isin(some_list)
```

# Returns a Series object filled with True or False by checking if the values of the column exists in the given list

---

```
df.groupby('column_name')
```

# Groups the whole dataframe into groups based on the values of the given column. You can then add .mean(), .min(), .count() and similar functions to calculate extra stats on these groups.

# One example is if you have a dataframe with heights of students in a class, by doing

```
df.groupby('gender').mean()
```

# you will have the mean height of each gender.

Extracting values from the dataframe

```
df.iloc[n]
```

# Gets you the row of the dataframe that is physically the nth row.

```
df.loc[n]
```

# gets you the nth row of the dataframe **but** based on the index of the dataframe. The index and the physical location might be different after you sort your dataframe or perform another action that effects the index. (see df.reset_index())

```
df[['column1', 'column2']]
```

# Returns the selected columns in a dataframe object. You can get as many columns as you want.

```
df['column_name']
```

# Returns the selected column in a Series object. You can only specify one column name in this version.

Quickly plotting the dataframe

```
df['column_name'].hist()
```

# Get a histogram for this column.

```
df.hist()
```

# Get a histogram for every column that is numerical.

```
df.plot()
```

# Returns a line chart with all the numerical columns plotted on the y axis, index on the x axis.

Dealing with duplicates and missing values

```
df.drop(number)
```

# Returns a dataframe where the row with the given number is removed.

```
df.drop('column_name', axis=1)
```

# Same as the previous function but removes the given column.

```
df.dropna()
```

# Returns a dataframe where all the rows that include a missing value is removed.

```
df.drop_duplicates()
```

# Returns a dataframe where rows with duplicate values is removed to only leave one copy.

```
df.drop_duplicates(['column_name'])
```

# Same as previous but only takes the given column to check for duplicates values. You can give it more than one column name.

```
df.fillna(value)
```

# Returns a dataframe where all the missing values are replaced with the given value.

```
df['column_name'].fillna(value)
```

# Same as the previous function but returns only the given column with the missing values filled with the value.

Tips:

My go-to way of finding more information on a certain pandas functions is simply Googling:
pandas <name of function>

And If I ever struggle with finding the exact function/ combination of functions to perform an action, I again simply Google the exact thing I want to do.

And I mean very simply...
Some examples are:

- *"pandas make columns into rows"*
- *"pandas remove second level of columns after merge"*
- *"pandas get count of unique values of columns"*

And 99.9% of the time I find exactly what I was looking for. So don't be afraid to go and look for answers online.

That's how I learned much of what I know today.

Hopefully this document will help you get a head start without having to go through years of Googling.

Enjoy!

— Mısra

Filter by value

```
df[df['column_name']==2]
```

# Only returns the rows where the value for the given column satisfies the condition. You can have any condition inside the brackets as long as it returns True and False in a Series object and it will work.

```
df[(df['column1']==2) & ~(df['column2']<10)]
```

# If you'd like to filter with multiple conditions, you can concatenate them by wrapping each in parentheses and merging them with "and" (&) and/or "or" (|) symbol. If you'd like to negate one of the conditions, add a tilde before the condition.

Functions involving two dataframes and supporting functions

```
pd.concat([df1,df2])
```

# Concatenate two dataframes end to end. One way to think of it is adding rows. Returns a dataframe.

```
df1.merge(df2, how='left', on='col_name')
```

# Merge two dataframes into one. You can think of this as adding columns. Returns a dataframe.

# It is possible to specify the type of merging (inner, outer, left, right). You also need to specify which column should be taken as the reference when merging.

```
df.copy()
```

# Returns a copy of the dataframe. This functions is used when you don't want to accidentally change the original dataframe while doing actions on it.

```
df.reset_index()
```

# Resets the index to be ordered. Returns a dataframe. This is needed when the index gets shuffled due to merging two dataframes or sorting a dataframe.