

# Pandas: Further Readings and Cheat Sheet

Pandas [official documentation](https://pandas.pydata.org/pandas-docs/stable/10min.html) is very extensive. To make navigating through it easier, here are some good places for a broader and/or more detailed overview:

- [Essential Basic Functionality](#)
- [Tutorials](#)
- [Cookbook](#)

Below is a handy **Pandas cheat sheet**!

## Data Wrangling with pandas Cheat Sheet

<http://pandas.pydata.org>

### Syntax - Creating DataFrames

```
df = pd.DataFrame(
    {'M': [7, 8, 9],
     'F': [7, 8, 9],
     'A': [7, 8, 9]},
    index=[1, 2, 3])
```

### Method Chaining

```
df = (pd.DataFrame(
    {'M': [7, 8, 9],
     'F': [7, 8, 9],
     'A': [7, 8, 9]}))
df['M'] = df['M'] + 1
```

### Tidy Data

Each variable is stored in its own column & Each observation is stored in its own row

Try to keep elements in tidy's **wide**bed operations as the tidy will automatically preserve observations as you manipulate variables. No other format can do so natively with pandas.

M \* A → F

### Reshaping Data - Change the layout of a dataset

**pd.melt(df)**  
Gather columns into rows

**df.pivot(columns='var', values='val')**  
Spread rows into columns

**pd.concat([df1, df2])**  
Append rows of DataFrames

**pd.concat([df1, df2], axis=1)**  
Append columns of DataFrames

**df.reset\_index()**  
Reset Index on DataFrame

**df.reset\_index(inplace=True)**  
Reset Index on DataFrame (inplace)

**df.sort\_values('var', ascending=False)**  
Sort DataFrame by values of 'var' in descending order

**df.sort\_index()**  
Sort DataFrame by index

**df.dropna()**  
Drop rows with missing values

**df.dropna(inplace=True)**  
Drop rows with missing values (inplace)

**df.dropna(axis=1, how='any')**  
Drop columns with missing values

**df.dropna(axis=1, how='all')**  
Drop columns with missing values (all)

### Subset Observations (Rows)

**df[df.length > 7]**  
Select rows where length is greater than 7

**df.sample(n=5)**  
Randomly select 5 rows

**df.sample(frac=.5)**  
Randomly select 50% of rows

**df.ix[10:20]**  
Select rows by position

**df.ix[10:20, 'length']**  
Select rows by position and column

**df.ix[10:20, ['length', 'weight']]**  
Select rows by position and columns

### Subset Variables (Columns)

**df[['length', 'weight', 'species']]**  
Select multiple columns and rows

**df[['length']] = df[['length']]**  
Selecting a column as a specific name

**df[['length']] = df[['length']]**  
Selecting a column as a specific name

**df[['length']] = df[['length']]**  
Selecting a column as a specific name

**df[['length']] = df[['length']]**  
Selecting a column as a specific name

## Summarize Data

```
df[["a"]].value_counts()
# Print number of rows with each unique value in column
len(df)
# All rows in DataFrame
df[["a"]].unique()
# All values in column 'a'
df.describe()
# Basic descriptive statistics for each column in DataFrame
```



perform multiple functions on many features (add, update, or delete) and apply and aggregate (reduce) on columns using GroupBy, Expanding and Rolling (see below) and produce single values for each of the groups. When applied to a DataFrame, the result is normally a pandas Series, unless the examples:

```
sum()
# Sum of each column
count()
# Count non-NaN values of each column
median()
# Median value of each column
quantile([0.25, 0.75])
# Quantile values of each column
apply(function)
# Apply function on each column
```

```
min()
# Minimum value of each column
max()
# Maximum value of each column
mean()
# Mean value of each column
var()
# Variance of each column
std()
# Standard deviation of each column
```

## Group Data



```
df.groupby("a")
# Group by column 'a', grouped values in column 'a' are in 'a'
```

```
df.groupby(["a", "b"])
# Group by columns 'a' and 'b', grouped values in columns 'a' and 'b' are in 'a' and 'b'
```

All of the summary functions that can be used on grouped data can be applied to grouped data using the following functions:

```
agg(function)
# Aggregate grouped data using function
```

## Windows

```
df.rolling(5)
# Return a DataFrame with 5 rows of data for each column in df
df.rolling(5).sum()
# Return a DataFrame with 5 rows of data for each column in df, applying the sum function to the data
```

## Handling Missing Data

```
df.dropna()
# Drop rows with one or more missing values
df.fillna(value)
# Replace all missing values with 'value'
```

## Make New Columns



```
df.assign(new_column=df["a"]*df["b"])
# Create a new column 'new_column' from existing columns 'a' and 'b'
df["volume"] = df["length"]*df["height"]*df["width"]
# Add a new column 'volume' from existing columns 'length', 'height', and 'width'
```



perform multiple functions on many features (add, update, or delete) and apply and aggregate (reduce) on columns using GroupBy, Expanding and Rolling (see below) and produce single values for each of the groups. When applied to a DataFrame, the result is normally a pandas Series, unless the examples:

```
max(axis=1)
# Maximum value of each row
min(axis=1)
# Minimum value of each row
clip(lower=0, upper=10)
# Clip values of each column to the range [0, 10]
```

The examples are used to demonstrate applying functions to the data using the function applied on a per-group basis, and the assumed action on all the rows of the original DataFrame.

```
shift(1)
# Shift values of each column by 1 row
shift(-1)
# Shift values of each column by -1 row
cumsum()
# Cumulative sum of each column
cummin()
# Cumulative minimum of each column
cummax()
# Cumulative maximum of each column
```

```
shift(1)
# Shift values of each column by 1 row
shift(-1)
# Shift values of each column by -1 row
cumsum()
# Cumulative sum of each column
cummin()
# Cumulative minimum of each column
cummax()
# Cumulative maximum of each column
```

## Plotting

```
df.plot()
# Plot each column in df as a line plot
df.plot.scatter(x="a", y="b")
# Scatter plot of column 'a' vs column 'b'
```



## Combine Data Sets

```
df1 + df2
# Add two DataFrames
```

df1

```
a 1
b 2
c 3
```

df2

```
a 2
b 3
c 4
```

=

```
pd.merge(df1, df2, how="left", on="a")
# Merge df1 and df2 on column 'a' using left join
```

```
pd.merge(df1, df2, how="right", on="a")
# Merge df1 and df2 on column 'a' using right join
```

```
pd.merge(df1, df2, how="inner", on="a")
# Merge df1 and df2 on column 'a' using inner join
```

```
pd.merge(df1, df2, how="outer", on="a")
# Merge df1 and df2 on column 'a' using outer join
```

```
df1[df1["a"] > 1]
# Filter df1 where column 'a' is greater than 1
```

```
df1[df1["a"] > 1 & df2["a"] > 1]
# Filter df1 and df2 where column 'a' is greater than 1
```

```
df1 + df2
# Add two DataFrames
```

df1

```
a 1
b 2
c 3
```

```
a 2
b 3
c 4
```

=

```
pd.merge(df1, df2, how="left", on="a")
# Merge df1 and df2 on column 'a' using left join
```

```
pd.merge(df1, df2, how="right", on="a")
# Merge df1 and df2 on column 'a' using right join
```

```
pd.merge(df1, df2, how="inner", on="a")
# Merge df1 and df2 on column 'a' using inner join
```

```
pd.merge(df1, df2, how="outer", on="a")
# Merge df1 and df2 on column 'a' using outer join
```