Optimus Cheat Sheet

Agile Data Science Workflows made easy.

Initialize Optimus

```
from optimus import Optimus
op = Optimus("pandas")
```

Import optimus

Creating DataFrames

```
df = op.create.dataframe({
    ("names"): ["bumbl#ebéé  ",
"Optim'us", "ironhide&"],
    ("height", "float"): [17.5, 28.0,
26.0],
    "function": ["Espionage",
"Leader", "Security"],
    ("rank", "int"): [7, 10, 7]
})
```

names	height	function	rank
bumbl#ebéé··	17.5	Espionage	7
Optim'us	28.0	Leader	10
ironhide&	26.0	Security	7

Specify values for each column.

Data Loading

```
df = op.load.csv("foo.csv")
df = op.load.json("foo.json")
df = op.load.parquet("foo.parquet")
df = op.load.excel("foo.xls")
df = op.load.avro("foo.avro")
df = op.load.file("foo.anything")
```

Load a file

```
df =
op.load.csv("http://.../foo.csv")
df = op.load.json("http://./f.json")
df =
op.load.file("http://./f.parquet")
```

Load a file to a dataframe from a URL

Data Saving

```
df.save.csv("directory/foo.csv")
df.save.json("directory/foo.json")
df.save.parquet("directory/foo.parquet

df.save.excel("directory/foo.xls")
df.save.avro("directory/foo.avro")
```

Save a local file

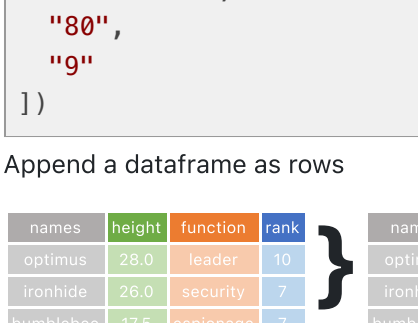
Method Chaining

Most Optimus methods return a DataFrame so that another Optimus method can be applied to the result. This improved readability of code.

Reshaping Data

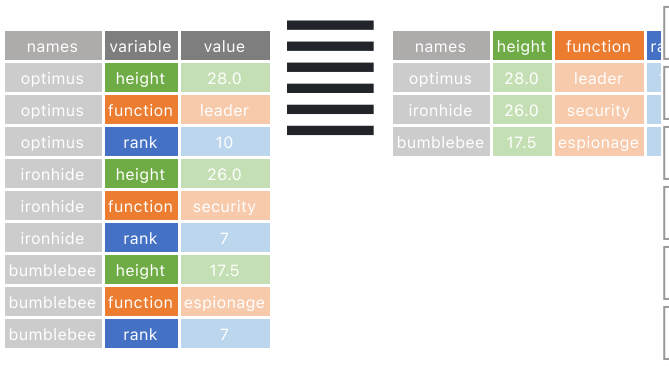
```
df = df.melt(
    "names",
    ["height", "function", "rank"])
```

Gather columns into rows.



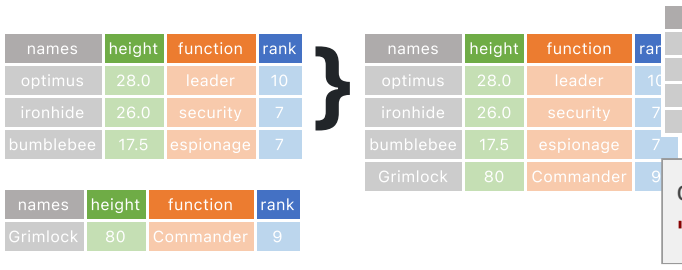

```
df = df.pivot("names", "variable",
"value")
```

Spread rows into columns.




```
df = df.rows.append([
  "Grimlock",
  "Commander",
  "80",
  "9"
])
```

Append a dataframe as rows




```
df = df.rows.sort("names")
```

Order rows by values of a column (low to high).

```
df = df.rows.sort("names", "asc")
```

Order rows by values of a column (high to low).

```
df = df.cols.rename("names", "name")
```

Rename the columns of a DataFrame.

```
df = df.cols.rename([
  ("name", "names"),
  ("function", "task")])
```

Drop columns from a DataFrame.

Handling Missing Data

```
df.rows.drop_na(cols)
```

Drop rows any column having null data.

```
df.cols.fill_na(cols, output_cols,
value)
```

Select Rows

```
df.display(n)
```

Show first n rows

```
df.rows.drop_duplicated()
```

Remove duplicate rows (only considers columns).

```
df.rows.sample(n)
```

Randomly select n rows

```
df.rows.select(df["rank"]>7)
```

Extract rows that meet logical criteria.

`df["A"] < df["B"]`	Less than
`df["A"] > df["B"]`	Greater than
`df["A"] == df["B"]`	Equal to
`df["A"] <= df["B"]`	Less than or equal to
`df["A"] >= df["B"]`	Greater than or equal to
`df["A"] != df["B"]`	Not equal to
`~df["A"]`	Negation
`df["A"] & df["B"]`	And
`df["A"] | df["B"]`	Or

Select Columns

```
df.cols.select(["names", "height",
"function"])
```

Select multiple columns with specific names.

```
df.cols.select([1, 3, 5])
```

Select columns in positions 1, 3 and 5 (first column is 0)

```
df.cols.select("n.*", regex=True)
```

Select columns whose name matched regular expression regex.

'\.	Matches strings containing a period
'Length$'	Matches strings ending wirt word 'Length'
'^$epal	Matches strings begining with the word 'Sepal'
'^x[1-5]$	Matches strings beginning with 'x' and ending with 1,2,3,4,5
''^(?!Species$).*'	Matches strings expect the string 'Species'

Unnest

```
df = df\
    .rows.sort(["rank", "height"])\
    .cols.lower(["names",
"function"])\
    .cols.normalize_chars("names")\

.cols.remove_special_chars("names")\
    .cols.trim("names")
```

names	height	function	rank
optimus	28.0	leader	10
ironhide	26.0	security	7
bumblebee	17.5	espionage	7

Summarize Data

Optimus provides a large set o summary functions.

```
df.cols.sum()
```

Sum all values in a column.

```
df.cols.min()
```

Min value in a column.

```
df.cols.max()
```

Max value in a column.

```
df.cols.median()
```

Median value in a column.

```
df.cols.mean()
```

Mean value in a column.

```
df.cols.std()
```

Standard Deviation in a column.

```
df.cols.quantile([0.25,0.75])
```

Qunatiles in a column.

Machine Learning

```
model =
df.ml.logistic_regression_text("senten
```

Create a model using Logistic Regression text.

```
model =
df.ml.random_forest("diagnosis")
```

Create a model using Random Forest.

```
model = df.ml.decision_tree(cols,
"diagnosis")
```

Create a model using Decision Tree.

```
model = df.ml.gbt(cols, "diagnosis")
```

Create a model using Gradient Boosting Trees.

Replace all null data with value.

```
df.cols.impute(cols, strategy="mean")
```

Replace all null data using a strategy.

Make new columns

```
df.cols.append("new_rank",
df["rank"]+"1")
```

Compute and append a new column.

```
df.cols.qcut("height", "bins", 2)
```

Bin columns into n buckets.

String Processing

Key Collision

```
df.cols.fingerprint(df, "names")
```

Create a fingerprint from a string.

```
df.string_clustering("names",
"fingerprint")
```

Cluster a dataframe column based on the fingerprint algorithm.

```
df.cols.n_gram_fingerprint(df,
"names", 2)
```

Calculate the 2-gram for a fingerprinted string.

```
df.string_clustering("names",
"n_gram_fingerprint", 2)
```

Cluster a DataFrame column based on the n-gram fingerprint algorithm.

```
df = df\
    cols.unnest("col_to_unnest")
```

Unnest a string, array or vector column

Nest

```
df = df\
    .cols.nest(["names", "function"],
     output_col = "nested_col",
     separator=" ")
```

Merge multiple columns as string

```
df = df
    .cols.nest(["names", "function"],
    output_col = "new_col",
    shape ="array")
```

Merge multiple columns as array

Plotting

```
df.plot.hist("*")
```

Histogram for all columns.

```
df.plot.frequency("*")
```

Frequency plot for all columns.

```
df.plots.correlation("*")
```

Correlation plot for all columns.

Profiling

```
df.profile("*")
```

Profile every column.

```
df.profile("names")
```

Profile a specific column.

```
df.profile(["names", "height"])
```

Profile multiple columns.