OBJECT ORIENTED PROGRAMMING IN PYTHON

Working with a DataSet to Create DataFrames

Vicki Boykis
Senior Data Scientist

MTCars

```
A data frame with 32 observations on 11 (numeric) variables.

[, 1]    mpg    Miles/(US) gallon
[, 2]    cyl    Number of cylinders
[, 3]    disp    Displacement (cu.in.)
[, 4]    hp    Gross horsepower
[, 5]    drat    Rear axle ratio
[, 6]    wt    Weight (1000 lbs)
[, 7]    qsec    1/4 mile time
[, 8]    vs    Engine (0 = V-shaped, 1 = straight)
[, 9]    am    Transmission (0 = automatic, 1 = manual)
[,10]    gear    Number of forward gears
[,11]    carb    Number of carburetors
```

model	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gea
Mazda RX4	21	6	160	110	3.9	2.62	16.46	0	1	4
Mazda RX4 Wag	21	6	160	110	3.9	2.875	17.02	0	1	4
Datsun 710	22.8	4	108	93	3.85	2.32	18.61	1	1	4

Creating our Cars analysis DataShell

```
# creating an instance of a DataShell
car_data = DataShell('mtcars.csv')
```

```
# print the object
print(car_data)
```

```
# The instance of the object
<__main__.DataShell object at 0x11090f8d0>
```

Creating a method to introspect the object

```
class DataShell:

    def __init__(self, filename):
        self.filename = filename

    def create_datashell(self):
        self.array = np.genfromtxt(self.filename, delimiter=',', dtype=None)
        return self.array

    def show_shell(self):
        print(self.array)
```

Printing the array

```
>>>>print(type(CarData.array))
  <class 'numpy.ndarray'>
```

```
>>>print(CarData.array)
[[b'model' b'mpg' b'cyl' b'disp' b'hp' b'drat' b'wt' b'qsec' b'vs' b'am'
  b'gear' b'carb']
 [b'Mazda RX4' b'21' b'6' b'160' b'110' b'3.9' b'2.62' b'16.46' b'0' b'1'
  b'4' b'4']
 [b'Mazda RX4 Wag' b'21' b'6' b'160' b'110' b'3.9' b'2.875' b'17.02' b'0'
  b'1' b'4' b'4']
 [b'Datsun 710' b'22.8' b'4' b'108' b'93' b'3.85' b'2.32' b'18.61' b'1'
  b'1' b'4' b'1']]
```

OBJECT ORIENTED PROGRAMMING IN PYTHON

Let's practice!

OBJECT ORIENTED PROGRAMMING IN PYTHON

Renaming Columns and the Five-Figure Summary

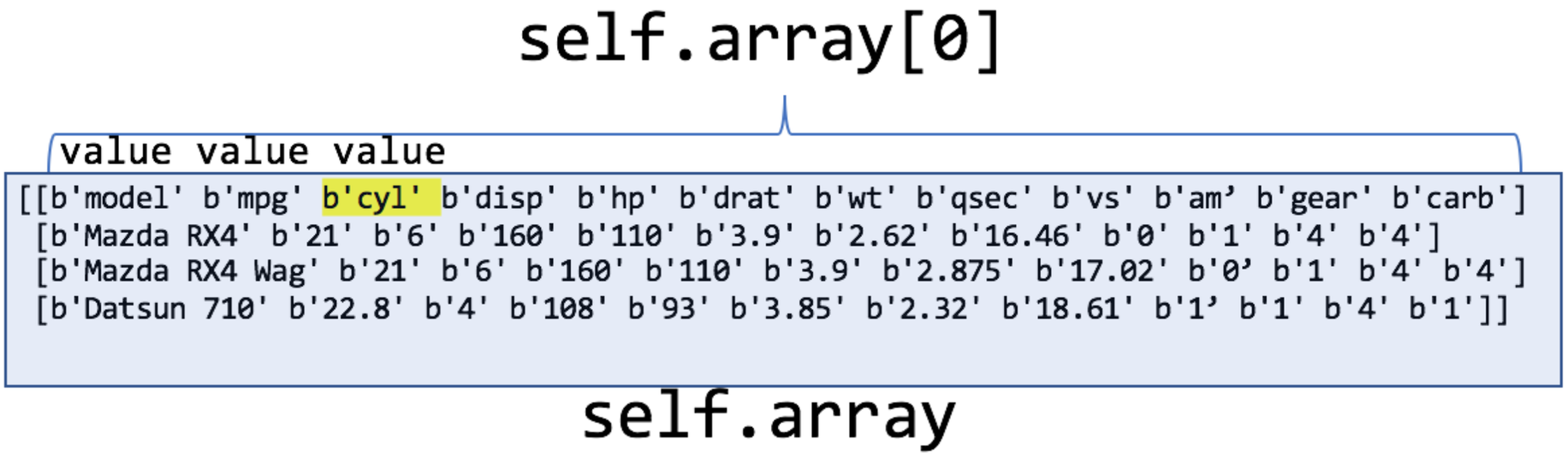
Vicki Boykis
Senior Data Scientist

Taking a second look at our column names

```
print(CarData.array)

[[b'model' b'mpg' b'cyl' b'disp' b'hp' b'drat' b'wt' b'qsec' b'vs' b'am'
  b'gear' b'carb']
 [b'Mazda RX4' b'21' b'6' b'160' b'110' b'3.9' b'2.62' b'16.46' b'0' b'1'
  b'4' b'4']
 [b'Mazda RX4 Wag' b'21' b'6' b'160' b'110' b'3.9' b'2.875' b'17.02' b'0'
  b'1' b'4' b'4']
 [b'Datsun 710' b'22.8' b'4' b'108' b'93' b'3.85' b'2.32' b'18.61' b'1'
  b'1' b'4' b'1']]
```

Accessing Column Names



Renaming the columns by passing in multiple parameters

```
class DataShell:

    def __init__(self, filename):
        self.filename = filename


    def rename_column(self, old_colname, new_colname):

            for index, value in enumerate(self.array[0]):
                if value == old_colname.encode('UTF-8'):
                    self.array[0][index] = new_colname

            return self.array
```

Completing the Rename

```
myDatashell.rename_column('cyl','cylinders')

print(myDatashell.array)

[[b'model' b'mpg' b'cylinders' b'disp' b'hp' b'drat' b'wt' b'qsec' b'vs' b'am'
  b'gear' b'carb']
 [b'Mazda RX4' b'21' b'6' b'160' b'110' b'3.9' b'2.62' b'16.46' b'0' b'1'
  b'4' b'4']
 [b'Mazda RX4 Wag' b'21' b'6' b'160' b'110' b'3.9' b'2.875' b'17.02' b'0'
  b'1' b'4' b'4']
 [b'Datsun 710' b'22.8' b'4' b'108' b'93' b'3.85' b'2.32' b'18.61' b'1'
  b'1' b'4' b'1']
```

Five-figure summary

```
def five_figure_summary(self):
    statistics =  stats.describe(self.array[1:,col_pos].astype(np.float))
    return f"Five-figure stats of column {col_position}: {statistics}"
```

- Note that `f"a"` prints the string `a` with `{b}` being able to reference the variable b.

```
>>>>myDatashell.five_figure_summary(1)
'Five-figure stats of column 1: DescribeResult(nobs=32, minmax=(10.4, 33.9),
mean=20.090625000000003, variance=36.32410282258064,
skewness=0.6404398640318834, kurtosis=-0.20053320971549793)'
```

OBJECT ORIENTED PROGRAMMING IN PYTHON

Let's practice!

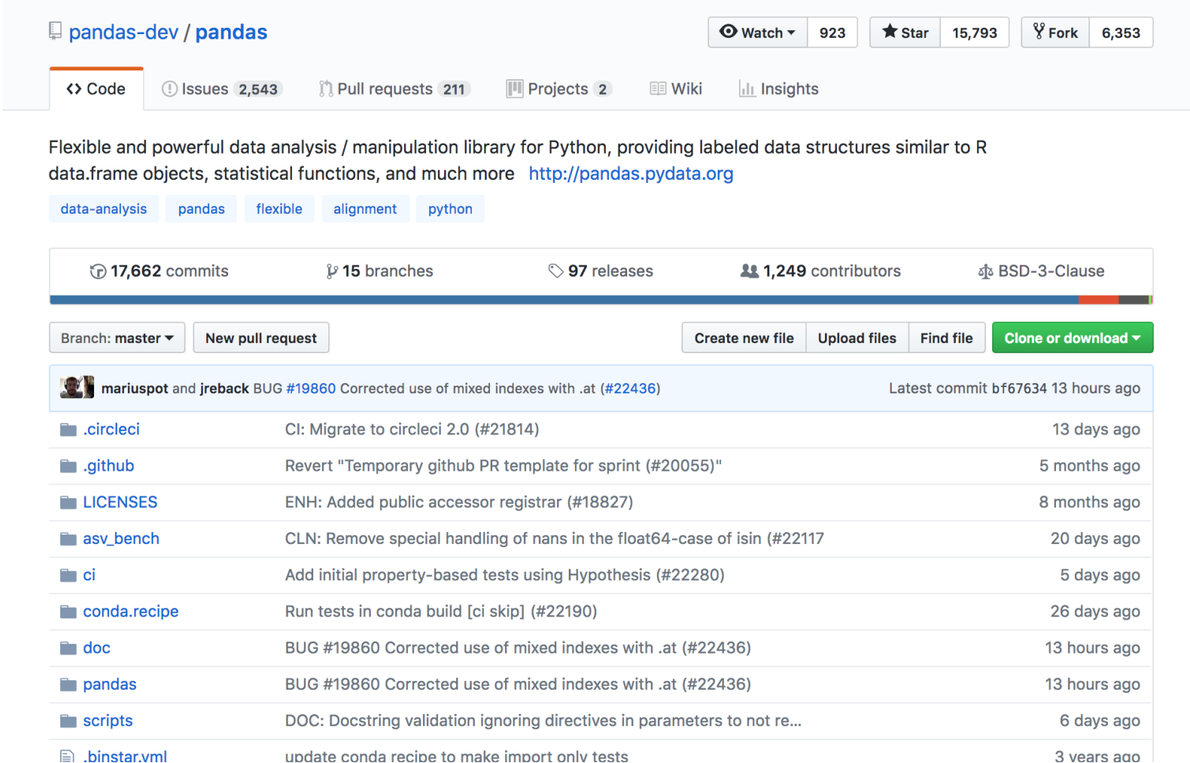
OBJECT ORIENTED PROGRAMMING IN PYTHON

Object Best Practices

Vicki Boykis
Senior Data Scientist

Reading Other People's Code

1. Check out GitHub Code.
2. Check out good examples of Python code:
3. Read the codebase.



Pandas and Spark

```
class BisectingKMeans(object):
    """
    A bisecting k-means algorithm based on the paper "A comparison of
    document clustering techniques" by Steinbach, Karypis, and Kumar,
    with modification to fit Spark.
    The algorithm starts from a single cluster that contains all points.
    Iteratively it finds divisible clusters on the bottom level and
    bisects each of them using k-means, until there are `k` leaf
    clusters in total or no leaf clusters are divisible.
    The bisecting steps of clusters on the same level are grouped
    together to increase parallelism. If bisecting all divisible
    clusters on the bottom level would result more than `k` leaf
    clusters, larger clusters get higher priority.

    Based on
    U{http://glaros.dtc.umn.edu/gkhome/fetch/papers/docclusterKDDTMW00.pdf}
    Steinbach, Karypis, and Kumar, A comparison of document clustering
    techniques, KDD Workshop on Text Mining, 2000.

    .. versionadded:: 2.0.0
    """

    @classmethod
    @since('2.0.0')
    def train(self, rdd, k=4, maxIterations=20, minDivisibleClusterSize=1.0, seed=-1888008604):
        """
        Runs the bisecting k-means algorithm return the model.

        :param rdd:
          Training points as an `RDD` of `Vector` or convertible
          sequence types.
        :param k:
          The desired number of leaf clusters. The actual number could
          be smaller if there are no divisible leaf clusters.
          (default: 4)
        :param maxIterations:
          Maximum number of iterations allowed to split clusters.
          (default: 20)
        :param minDivisibleClusterSize:
          Minimum number of points (if >= 1.0) or the minimum proportion
          of points (if < 1.0) of a divisible cluster.
          (default: 1)
        :param seed:
          Random seed value for cluster initialization.
          (default: -1888008604 from classOf[BisectingKMeans].getName.##)
        """
        java_model = callMLlibFunc(
            "trainBisectingKMeans", rdd.map(_convert_to_vector),
            k, maxIterations, minDivisibleClusterSize, seed)
        return BisectingKMeansModel(java_model)
```

Spark Class: The Class

```
class DataFrame(object):
    """A distributed collection of data grouped into named columns.
    A :class:`DataFrame` is equivalent to a relational table in Spark SQL,
    and can be created using various functions in :class:`SparkSession`::
        people = spark.read.parquet("...")
    Once created, it can be manipulated using the various
    domain-specific-language
    (DSL) functions defined in: :class:`DataFrame`, :class:`Column`.
    To select a column from the data frame, use the apply method::
        ageCol = people.age
    A more concrete example::
        # To create DataFrame using SparkSession
        people = spark.read.parquet("...")
        department = spark.read.parquet("...")
        people.filter(people.age > 30)
        .join(department, people.deptId == department.id) \\
          .groupBy(department.name, "gender")
          .agg({"salary": "avg", "age": "max"})
    .. versionadded:: 1.3
    """
```

Spark Class: The Constructor

```
def __init__(self, jdf, sql_ctx):
    self._jdf = jdf
    self.sql_ctx = sql_ctx
    self._sc = sql_ctx and sql_ctx._sc
    self.is_cached = False
    self._schema = None  # initialized lazily
    self._lazy_rdd = None
    # Check whether _repr_html is supported or not, we use it to avoid calling _
    # by __repr__ and _repr_html_ while eager evaluation opened.
    self._support_repr_html = False
```

Spark Class: A Method

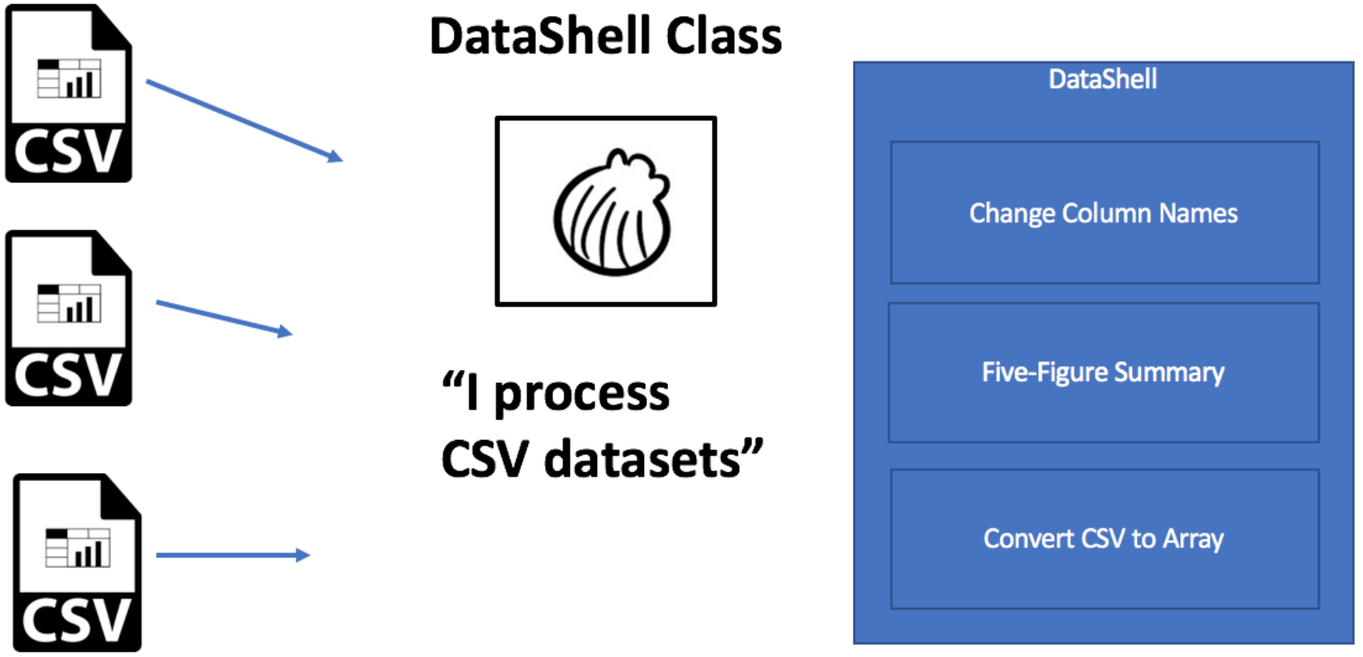
```
def printSchema(self):
        """Prints out the schema in the tree format.
        >>> df.printSchema()
        root
         |-- age: integer (nullable = true)
         |-- name: string (nullable = true)
        <BLANKLINE>
        """
        print(self._jdf.schema().treeString())
```

PEP Style

PEP 8 -- Style Guide for Python Code

PEP:	8
Title:	Style Guide for Python Code
Author:	Guido van Rossum <guido at python.org>, Barry Warsaw <barry at python.org>, Nick Coghlan <ncoghlan at gmail.com>
Status:	Active
Type:	Process
Created:	05-Jul-2001
Post-History:	05-Jul-2001, 01-Aug-2013

Separation of Concerns



OBJECT ORIENTED PROGRAMMING IN PYTHON

Let's practice!