Exploring Data with Python

Python

Python is a general-purpose, open-source computer programming language that has become one of the most popular programming languages in the data world.

```
# Python says Hello!
print("Hello World!")
```

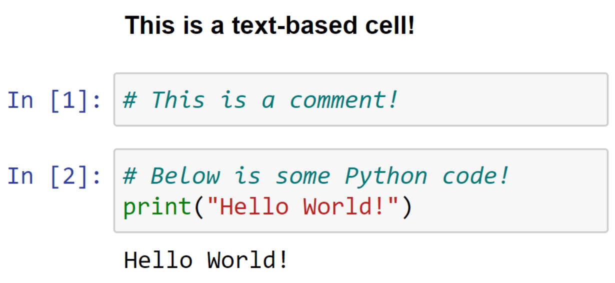
Jupyter Notebooks

Jupyter Notebooks are workspaces for interactively developing data science code in Python and other languages. They can be loaded directly in a web browser, and have cells/sections for

- creating documentation
- writing and running code
- displaying output and visualizations

Jupyter Notebook Cells

Jupyter Notebooks consist of sequential **cells**, where each cell contains either code, text-based comments, or special notebook commands. The bracketed numbers before the cells indicate the order in which the cells have been executed, which does not have to follow the order of the cells within the notebook. Any code output is displayed below the cell.



This is a text-based cell!

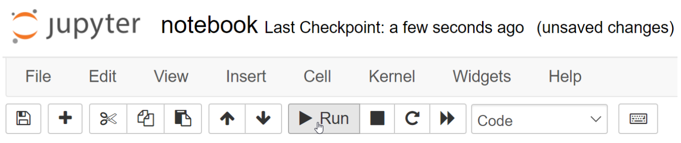
```
In [1]: # This is a comment!

In [2]: # Below is some Python code!
        print("Hello World!")

        Hello World!
```

Running a Jupyter Notebook cell

A cell in a Jupyter Notebook is run by either

1. selecting `Shift+Enter/Return` on the keyboard
2. selecting the Run button within the Notebook interface

If the code in the cell produces output, that output is displayed below the cell after the cell is run.


jupyter notebook Last Checkpoint: a few seconds ago (unsaved changes)
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Run
Code

Pandas

Pandas is a Python library for data analysis that comes with pre-packaged code for working with tables of data organized into rows and columns.

Import Pandas

Pandas is usually imported into a Python script using the alias `pd`.

```
import pandas as pd
```

CSV Files

Datasets are often stored in **comma-separated values** or **CSV** files, which store tables in plain text format, using commas to separate the columns of the dataset.
The example CSV in the code snippet for this review card produces the following table:

country	product_category	brand
gbr	aircon/ dehumidifier	delong
nld	kettle	royal swiss

```
country,product_category,brand
gbr,aircon/dehumidifier,delonghi
nld,kettle,royal swiss
```

codecademy

Importing CSVs with Pandas

The pandas method `.read_csv()` imports a CSV file as a pandas **DataFrame**.

```
# import the file 'dataset.csv'
# and assign it the name 'df'
df = pd.read_csv('dataset.csv')
```

Previewing a Pandas DataFrame

The DataFrame method `.head()` displays the first five rows of a DataFrame.

```
df.head()
```

Different Types of Data

Columns of a dataset can contain different types of data, like numbers, text, categories, or dates. Each type of data comes with different analytical questions and tools.

numeric	text	categories
12	marbles	toys
34	sheets of paper	office supplies

Pandas Column Data Types

Each column in a pandas DataFrame is automatically assigned a single data type. These include

- `int64` for integers (e.g. 1, -2, 0)
- `float64` for decimals (e.g. 3.14, 3.0)
- `object` for text

The DataFrame attribute `.dtypes` displays the datatype for each column.

```
# View all the data types in a DataFrame
df.dtypes
```

codecademy

Categorical Data Type

A column in a dataset is **categorical** if its values come from a small set of predetermined values (called **categories**).
For example, a `size` column where each entry is either `small`, `medium`, or `large` is categorical.
On the other hand, a `size` column where each entry is a measurement like weight or length is likely not categorical: there are many possible values each entry could have.

Python Variables

In Python, variables are used to store data. Variables are assigned data values with an equals sign (`=`):

```
variable = value
```

The value of a variable can be updated later:

```
variable = new_value
```

Variables can be named using a combination of numbers, letters, and underscores (`_`). They cannot start with a number.

```
# Define a variable months with value 11
months = 11

# Update months' value to 12
months = 12
```

Boolean Variables

In Python, a variable is **Boolean** if it has the value `True` or `False` (without quotes).

```
# Example Booleans
is_raining = True
is_sunny = False
```

codecademy

Python Data Types

Python data types include:

- int for integers
- float for decimals
- str for text strings
- bool for Booleans

The type() function outputs the type of a variable.

```
# Example int
number = 100

# Example float
score = 95.5

# Example str
color = 'red'

# Example bool
subscribed = True

# Display data type
print(type(color))
# Output: <class 'str'>
```

String Variables

In Python, a string (type str) is a sequence of characters (including numbers and special characters) surrounded by either single quotes 'string' or double quotes "string" .

```
# Example strings
single_quotes = 'Hello World!'
double_quotes = "Hello World!"
```

Pandas DataFrames

Datasets imported into (or created in) pandas have the Python variable type of DataFrame .
In its most basic form, a DataFrame contains data organized into rows and columns. An individual column of a DataFrame has the Python variable type Series .

codecademy

Jupyter Notebook Outputs

In a single Jupyter notebook cell, code is executed sequentially line-by-line from top to bottom. Typing a variable on the last line of a code cell will instruct the notebook to display that variable as output directly below the cell when it is run. To display more than one output, use `print()` around the extra variables to output.

```
variable1 = 3.14
variable2 = 100

print(variable1)
variable2
# Output:
# 3.14
# 100
```

Python Lists

In Python, a **list** is an ordered collection of values. Lists begin with an opening square bracket `[` and end with a closing square bracket `]`. Inside the brackets, the individual items in the list are separated by commas. These items can have any type.

```
# A list of strings
colors = ['red','green','blue']

# A list of floats
scores = [100.0, 95.54, 78.2]

# A list with varying types
random_list = [3.0, 'red']
```

Python Lists: .append()

The Python list method `.append()` adds a single item to the end of a list.

```
# Add the color 'orange' to the list of
colors
colors = ['red','green','blue','yellow']
colors.append('orange')
print(colors)
# Output:
['red','green','blue','yellow',
'orange']
```

Selecting an Item in a Python List

The items in a Python list are accessed by a built-in index that points to where the item is in the list (first, second, etc.).
To access an item in a list, place its index in square brackets.
Python uses **0-based** indexing: the first item is index 0 , the second item index 1 , and so on.

```
colors = ['red','green','blue','yellow']

# Access the color 'green' in colors
print(colors[1])
# Output:
# green
```

Python Dictionaries

In Python, a **dictionary** is a variable type that stores data as $key:value$ pairs. The key and value of each pair are separated by a colon (:), and the pairs are separated from each other by commas (,).

```
current_workspace = {
        'Language': 'python',
        'Development environment':
'jupyter',
        'Library': 'pandas'}
```

Accessing Python Dictionaries

A value in a Python dictionary can be accessed using bracket notation and the corresponding key: $dictionary[key]$.

```
repair = {
        'country':'swe',
        'product_category': 'mobile',
        'brand': 'apple',
        'year_of_manufacture': 2015.0}

# Access the brand of the repair
dictionary
print(repair['brand'])
# Output:
# apple
```

Updating Python Dictionaries

A new key:value pair can be added to a Python dictionary using the assignment operator (=)

```
dictionary[new_key] = new_valu
```

If the key already exists in the dictionary, the same syntax will update the value:

```
dictionary[old_key] = new_valu
```

```
repair = {
        'country':'swe',
        'product_category': 'mobile',
        'brand': 'apple',
        'year_of_manufacture': 2015.0}

# Assign a new key-value pair
repair['year_repaired'] = 2018

# Reassign the value of
'product_category' to 'mobile phone'
repair['product_category'] = 'mobile
phone'
```

Python Variable Methods

In Python, variables may come with built-in tools called **methods**.
A method is called by stating the variable name followed by a period (.), the method , and lastly parentheses () :

```
variable_name.method()
```

```
# .head() is a method to print five
lines of a dataset
df.head()

# .append() is a method for adding an
item to a list
colors = ['red','green','blue']
colors.append('orange')
```

Python Methods: Keyword Parameters

Python methods can have **keyword parameters** that modify the behavior of the method. Parameters are placed between the parentheses () after the method name is called using the following syntax:

```
variable_name.method(keyword=a
```

The argument is a value we select that alters the behavior of the method.

Previewing Data

By default, the DataFrame method .head() displays the first 5 rows of a DataFrame. Passing another number to the keyword n= alters the number of rows displayed.

```
# Output the first 5 rows of a DataFrame
df.head()

# Output the first 10 rows of a
DataFrame
# keyword: n
# argument: 10
df.head(n=10)
```

DataFrame Columns

One or more columns of a DataFrame can be selected using square brackets:

- df['column_1'] returns the data in column_1 as a Series
- df[['column_1']] returns the data in column_1 as a DataFrame
- df[['column_1', 'column_2']] returns the data in both column_1 and column_2 as a DataFrame

codecademy

.value_counts()

The method .value_counts() counts the number of times different values appear in a column of a DataFrame.
The code snippet in this review card demonstrates .value_counts() on the column repair_status below, from a DataFrame named repair .

repair_status
end of life
fixed
repairable
fixed
end of life

```
repair['repair_status'].value_counts()
# Output:
#       end of life    2
#       fixed          2
#       repairable     1
```

Percentage counts

When using .value_counts() on a column, passing True to the keyword normalize will return a percentage for each value in the column, instead of a raw count.
The code snippet in this review card applies normalize = True to the column below, from a DataFrame named repair . For example, 40% of the rows contain end of life .

repair_status
end of life
fixed
repairable
fixed
end of life

```
repair['repair_status'].value_counts(nor
malize=True)
# Output:
#       end of life    0.4
#       fixed          0.4
#       repairable     0.2
```

codecademy

Sorting .value_counts()

By default, the Pandas method `.value_counts()` is sorted from the most common value in a column to the least common value in the column.

Passing `True` to the `ascending` keyword reverses this order. The code snippet in this review card references the column below, from a DataFrame named `repair`.

repair_status
end of life
fixed
repairable
fixed
end of life

```
repair['repair_status'].value_counts(asc
ending=True)
# Output:
#       repairable     1
#       end of life    2
#       fixed          2
```

Pandas .describe() Method

The pandas method `.describe()` computes summary information of a Series/DataFrame.

On numeric columns, it returns the:

- `count` of all the non-missing numbers
- `mean` and `std` (standard deviation) of the numbers
- `min` and `max` value of the numbers
- `25th`, `50th`, `75th` percentiles of the numbers

On `object` (text) columns, it returns the:

- `count` of non-missing entries
- number of `unique` values
- most frequent/ `top` value
- `freq` : number of times the top value appears

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

Print   Share