Good Programming Practices: Standard Libraries

Analytics Vidhya

Good Programming Practices

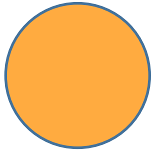
● Easy to Read ● Easy to Understand	● Documentation and Formatting
● Robust to Errors	● Testing and Debugging
● Robust to Unknown Use Cases	● Assertions and Exception Handling
● Code Reusability	● Standard Libraries

Good Programming Practices

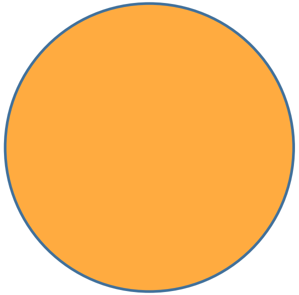
- Easy to Read
- Easy to Understand
- Robust to Errors
- Robust to Unknown Use Cases
- Code Reusability

- Documentation and Formatting
- Testing and Debugging
- Assertions and Exception Handling
- Standard Libraries

Recap: Functions

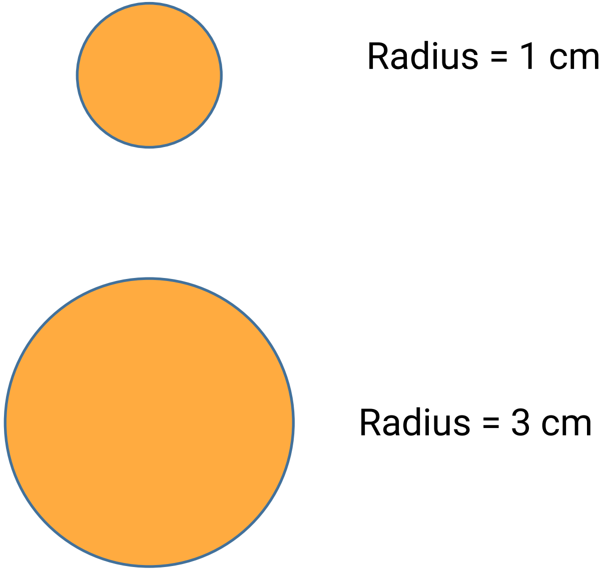


Radius = 1 cm



Radius = 3 cm

Recap: Functions

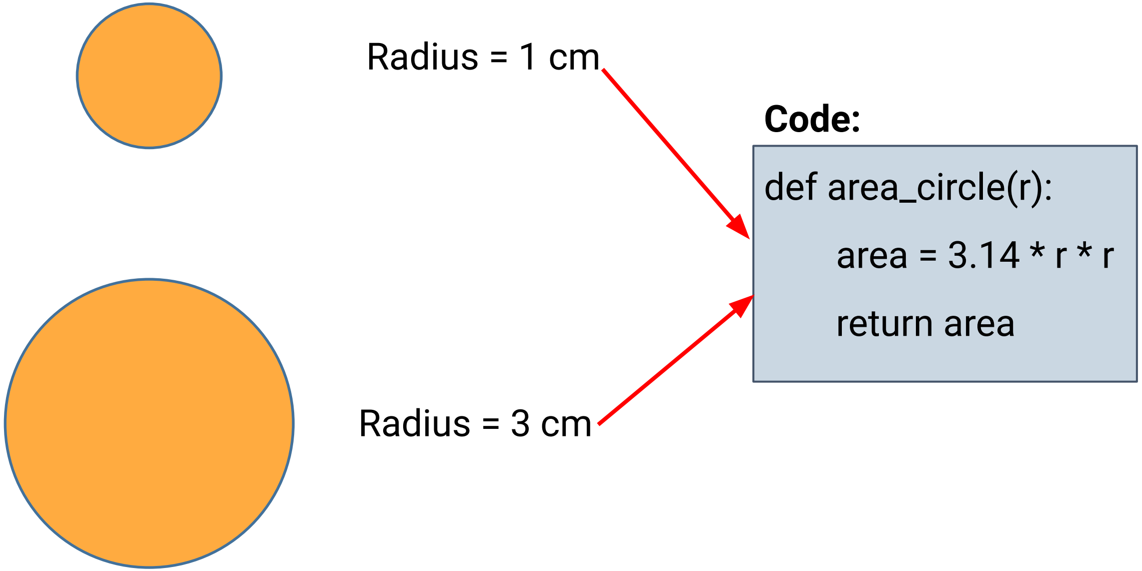


Radius = 1 cm

Radius = 3 cm

AREA?

Recap: Functions



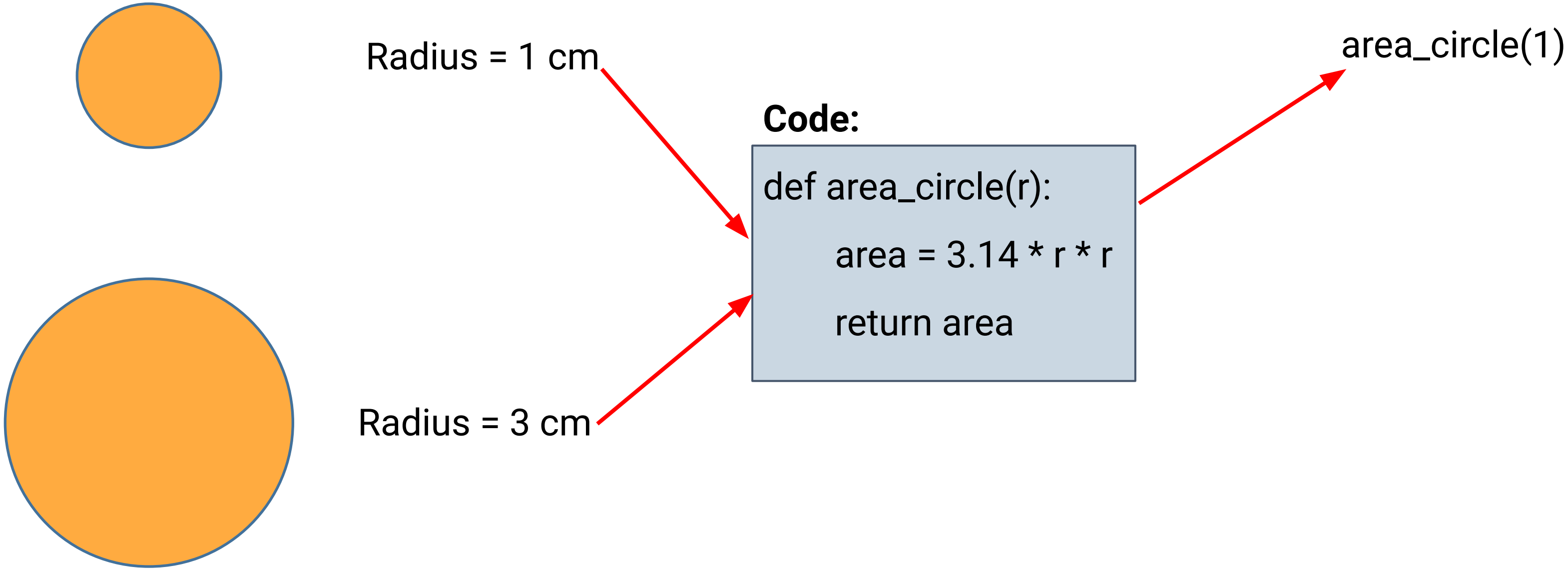
Radius = 1 cm

Radius = 3 cm

Code:

```
def area_circle(r):
        area = 3.14 * r * r
        return area
```

Recap: Functions



Radius = 1 cm

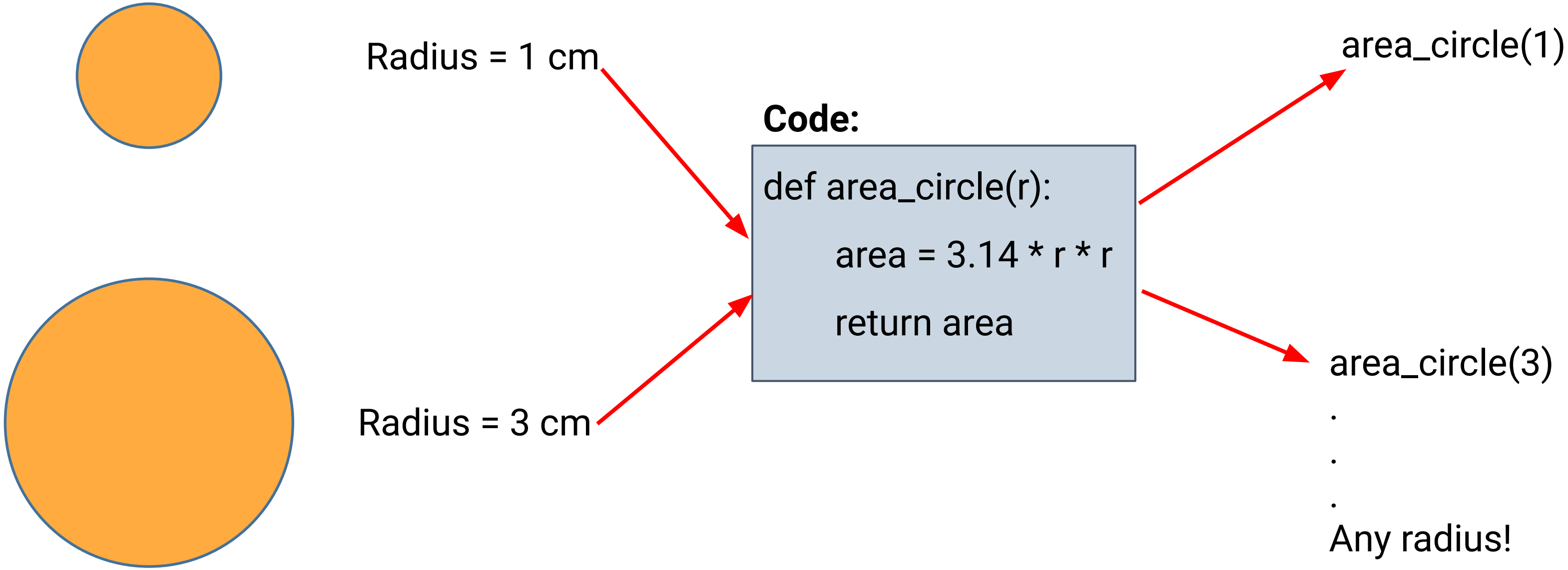
Radius = 3 cm

Code:

```
def area_circle(r):
       area = 3.14 * r * r
       return area
```

area_circle(1)

Recap: Functions



Radius = 1 cm

Radius = 3 cm

Code:

```
def area_circle(r):
        area = 3.14 * r * r
        return area
```

area_circle(1)

area_circle(3)

.

.

.

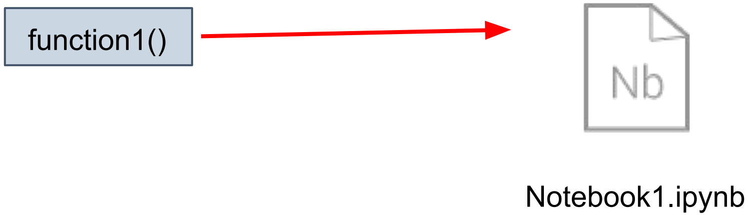
Any radius!

Recap: Functions

Nb

Notebook1.ipynb

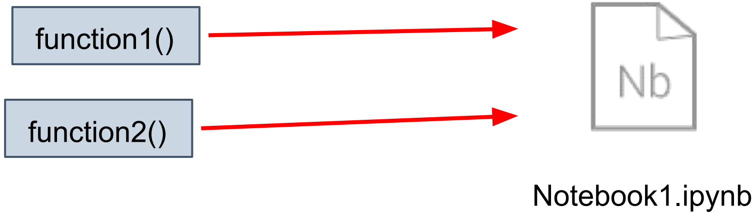
Recap: Functions



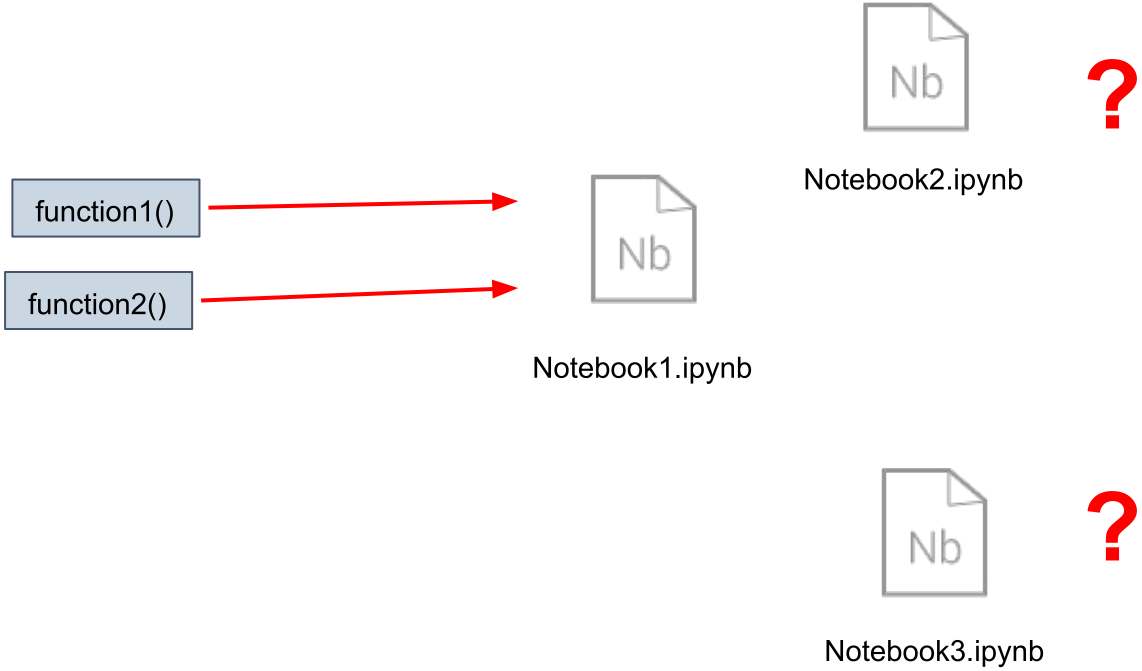
function1()

Notebook1.ipynb

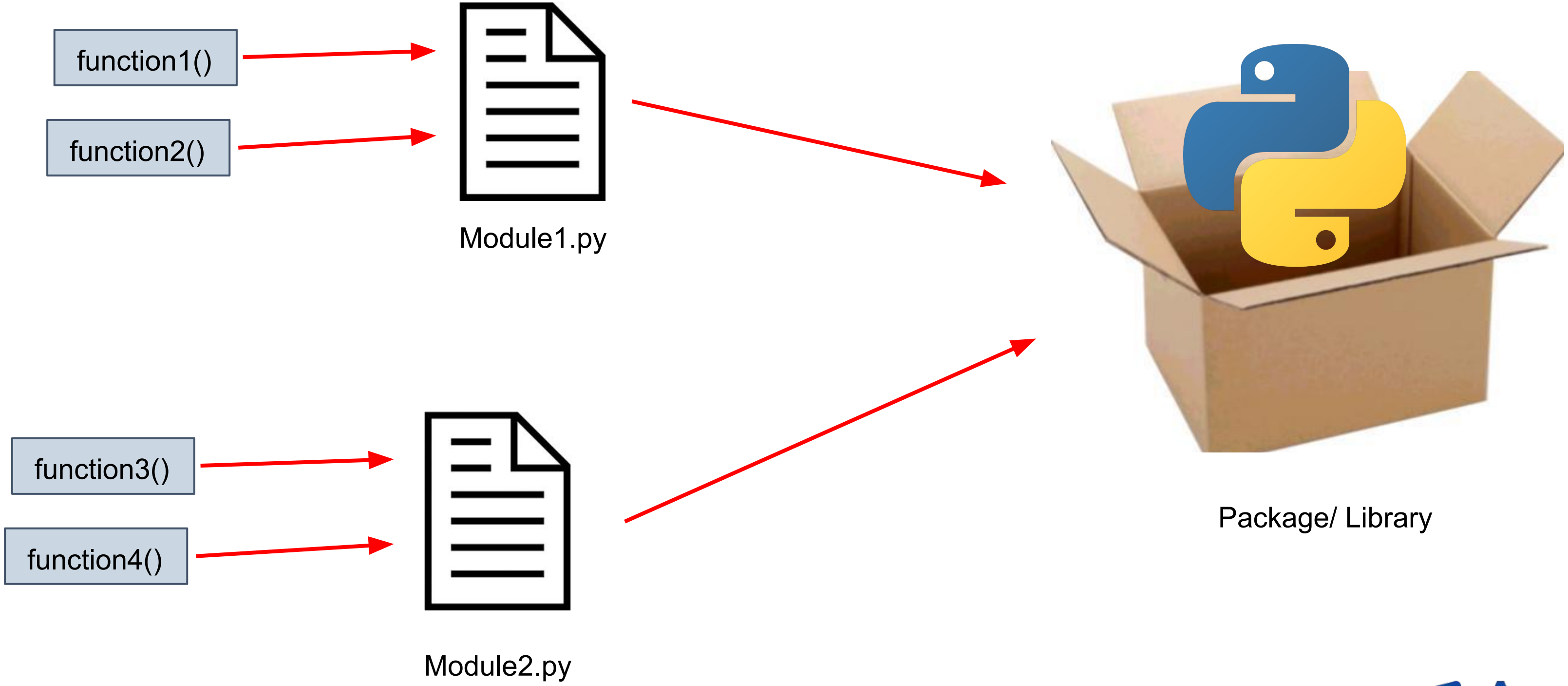
Recap: Functions



Recap: Functions



Standard Libraries and Modules in Python



Analytics Vidhya

Standard Libraries and Modules in Python

Package/ Library

Module.py

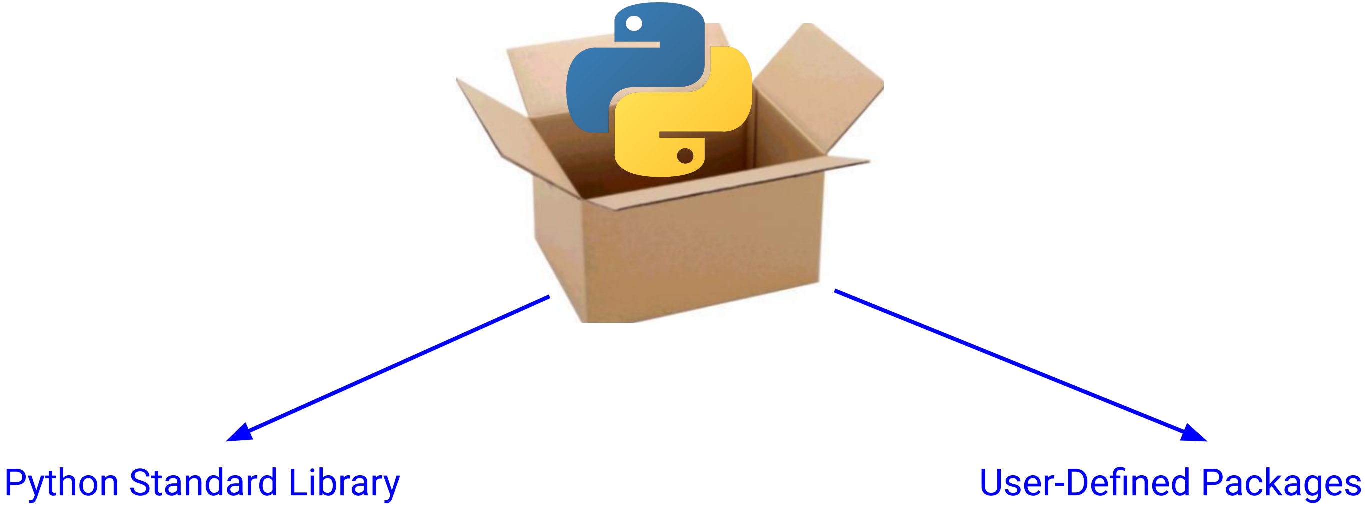
function()

Sklearn

sklearn.metrics

accuracy_score()

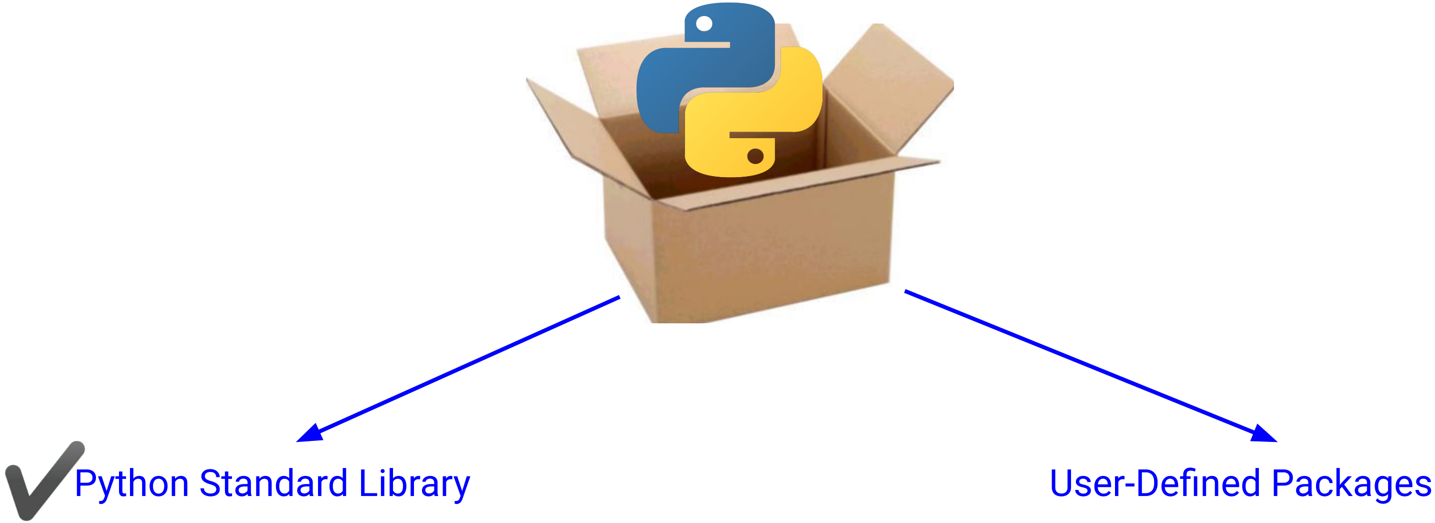
Standard Libraries and Modules in Python



Python Standard Library

User-Defined Packages

Python Standard Libraries



✓ Python Standard Library

User-Defined Packages

Python Standard Libraries

Python Standard Library is a collection of script modules accessible to a Python program to simplify the programming process and removing the need to rewrite commonly used commands.

Python Standard Libraries

We have already seen the following modules in the Python Basic Course

- math
- random
- datetime
- os

Python Standard Libraries

In this module we will cover the following standard libraries:

- itertools
- functools
- collections
- pickle

Complete List: The Python Standard Library

Thank You

Standard Libraries: itertools

Python Standard Libraries

In this module we will cover:

- itertools
- functools
- collections
- pickle

Complete List: The Python Standard Library

Python Standard Library: Itertools

- Collection of functions creating iterators for efficient looping

Python Standard Library: Itertools

- Collection of functions creating iterators for efficient looping
- Some of the popularly used functions are:
  - **filterfalse():** returns elements of seq where condition is false

Python Standard Library: Itertools

Income values:

10,000	22,100	12,000	7,000	15,000	90,000	45,000

Condition: Income<20,000

Python Standard Library: Itertools

Income values:

10,000	22,100	12,000	7,000	15,000	90,000	45,000

Condition: Income<20,000

Filterfalse Output: [22100, 90000, 45000]

Python Standard Library: Itertools

- Collection of functions creating iterators for efficient looping
- Some of the popularly used functions are:
  - **filterfalse():** elements of seq where condition is false
  - **permutation():** returns all possible orderings

Python Standard Library: Itertools

Different Cities

Noida	Gurugram	Delhi	Agra

Python Standard Library: Itertools

Different Cities

Noida	Gurugram	Delhi	Agra

Start - Destination

Permutation (repeat =2): N-G, N-D, N-A, G-N, G-D, G-A, ...

Python Standard Library: Itertools

Different Cities

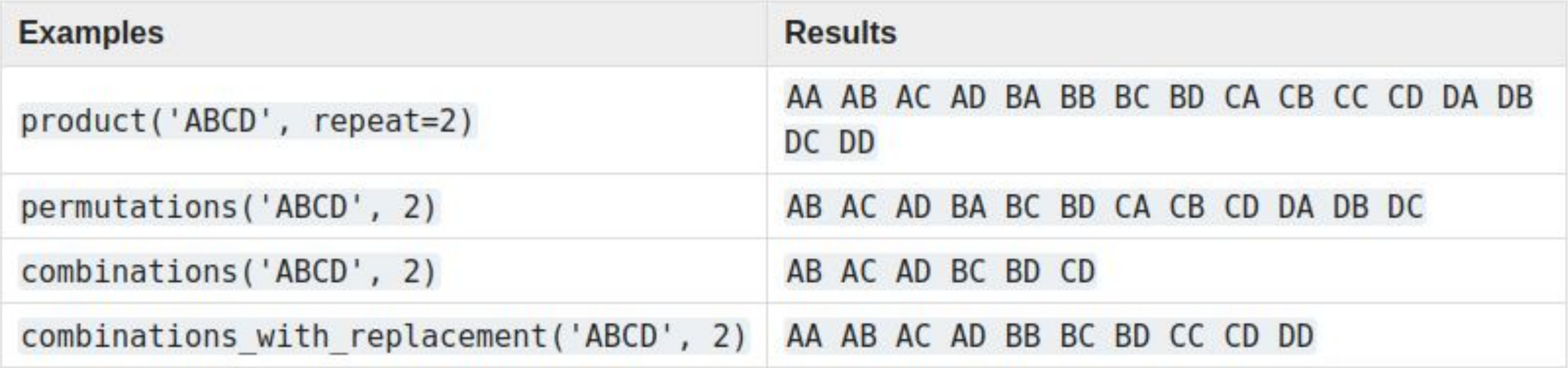
Noida	Gurugram	Delhi	Agra

Start - Destination

Permutation (repeat =2): N-G, N-D, N-A, G-N, G-D, G-A, ...

Permutation (repeat =3): N-G-D, N-G-A, N-D-G, N-D-A, ...

Python Standard Library: Itertools

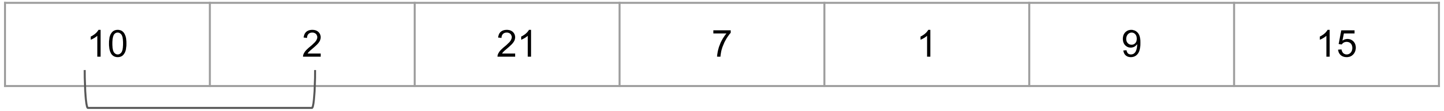


Examples	Results
`product('ABCD', repeat=2)`	`AA AB AC AD BA BB BC BD CA CB CC CD DA DB DC DD`
`permutations('ABCD', 2)`	`AB AC AD BA BC BD CA CB CD DA DB DC`
`combinations('ABCD', 2)`	`AB AC AD BC BD CD`
`combinations_with_replacement('ABCD', 2)`	`AA AB AC AD BB BC BD CC CD DD`

Python Standard Library: Itertools

- Collection of functions creating iterators for efficient looping
- Some of the popularly used functions are:
  - **filterfalse():** elements of seq where condition is false
  - **product():** returns all possible orderings
  - **accumulate():** returns accumulated results

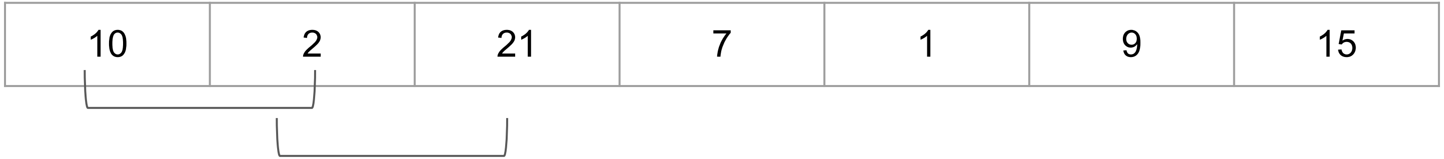
Python Standard Library: Itertools



10	2	21	7	1	9	15

Accumulated Sum: (10+2)

Python Standard Library: Itertools

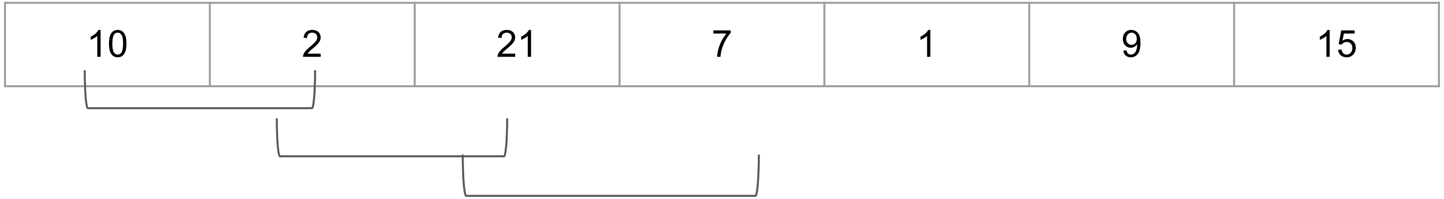


10	2	21	7	1	9	15

Accumulated Sum: (10+2)

Accumulated Sum: (12+21)

Python Standard Library: Itertools



10	2	21	7	1	9	15

Accumulated Sum: (10+2)

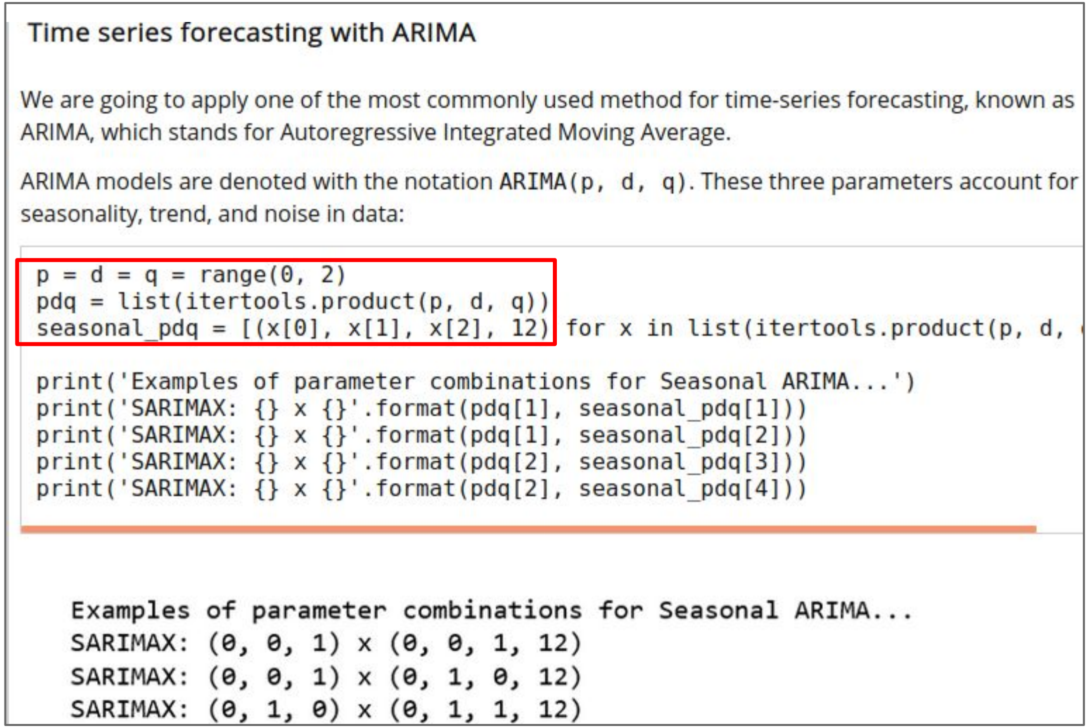
Accumulated Sum: (12+21)

Accumulated Sum: (33+7) … … …

Notebook

Applications: itertools

An End-to-End Project on Time Series Analysis and Forecasting



Time series forecasting with ARIMA

We are going to apply one of the most commonly used method for time-series forecasting, known as ARIMA, which stands for Autoregressive Integrated Moving Average.

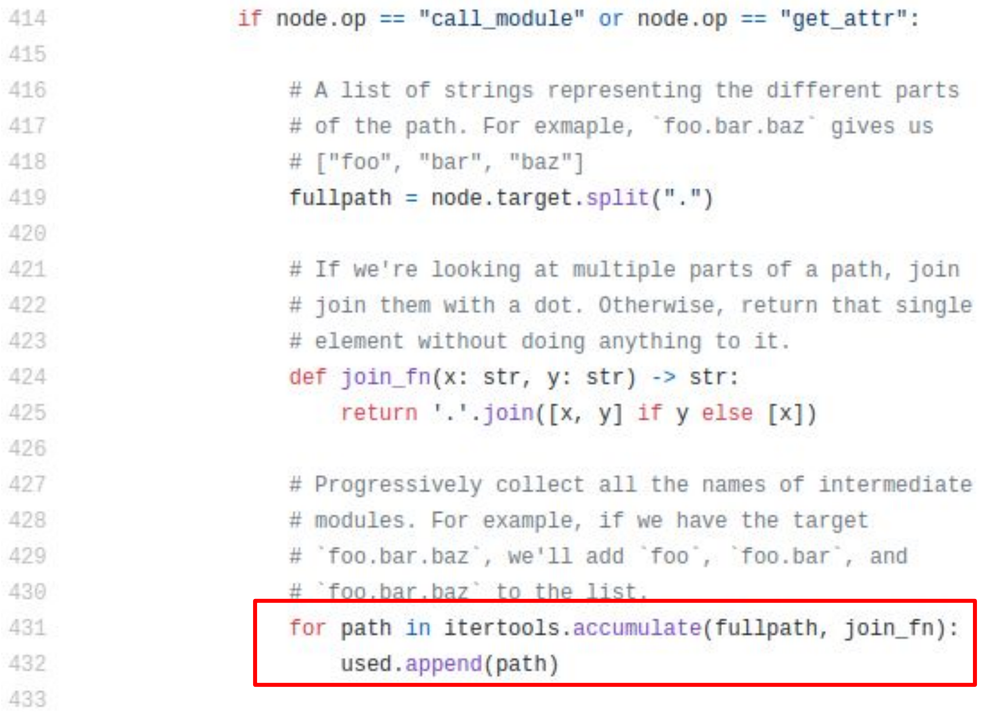
ARIMA models are denoted with the notation `ARIMA(p, d, q)`. These three parameters account for seasonality, trend, and noise in data:

```
p = d = q = range(0, 2)
pdq = list(itertools.product(p, d, q))
seasonal_pdq = [(x[0], x[1], x[2], 12) for x in list(itertools.product(p, d,

print('Examples of parameter combinations for Seasonal ARIMA...')
print('SARIMAX: {} x {}'.format(pdq[1], seasonal_pdq[1]))
print('SARIMAX: {} x {}'.format(pdq[1], seasonal_pdq[2]))
print('SARIMAX: {} x {}'.format(pdq[2], seasonal_pdq[3]))
print('SARIMAX: {} x {}'.format(pdq[2], seasonal_pdq[4]))
```

```
Examples of parameter combinations for Seasonal ARIMA...
SARIMAX: (0, 0, 1) x (0, 0, 1, 12)
SARIMAX: (0, 0, 1) x (0, 1, 0, 12)
SARIMAX: (0, 1, 0) x (0, 1, 1, 12)
```

Applications: itertools

PyTorch graph module



```python
if node.op == "call_module" or node.op == "get_attr":

    # A list of strings representing the different parts
    # of the path. For exmaple, `foo.bar.baz` gives us
    # ["foo", "bar", "baz"]
    fullpath = node.target.split(".")

    # If we're looking at multiple parts of a path, join
    # join them with a dot. Otherwise, return that single
    # element without doing anything to it.
    def join_fn(x: str, y: str) -> str:
        return '.'.join([x, y] if y else [x])

    # Progressively collect all the names of intermediate
    # modules. For example, if we have the target
    # `foo.bar.baz`, we'll add `foo`, `foo.bar`, and
    # `foo.bar.baz` to the list.
    for path in itertools.accumulate(fullpath, join_fn):
        used.append(path)
```

Thank You

Analytics Vidhya

Standard Libraries: functools

Python Standard Libraries

In this module we will cover:

- Itertools
- functools
- collections
- pickle

Complete List: [The Python Standard Library]()

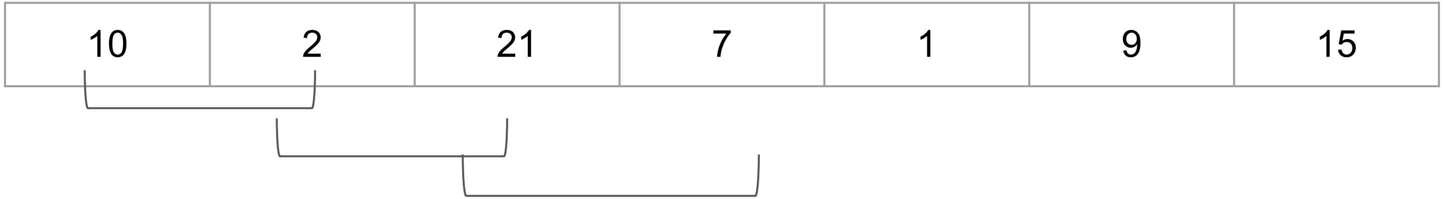
Python Standard Library: functools

- Higher-order functions and operations
- Returns a function or takes another function as an argument

Python Standard Library: functools

- Higher-order functions and operations
- Returns a function or takes another function as an argument
    - **reduce():** reducing iterable to a single cumulative value

Python Standard Library: functools



10	2	21	7	1	9	15

Accumulated Sum: (10+2)

Accumulated Sum: (12+21)

Accumulated Sum: (33+7) … … …

Reduce : 65

Python Standard Library: functools

- Higher-order functions and operations
- Returns a function or takes another function as an argument
    - **reduce():** reducing iterable to a single cumulative value
    - **lru_cache():** memoizing callable and returns the stored value

Python Standard Library: functools

Calculate 5!

1! = 1

2! = 1x2

3! = 1x2x3

4! = 1x2x3x4

.

.

Python Standard Library: functools

Calculate 5!

1! = 1

2! = 1x2

3! = 1x2x3

4! = 1x2x3x4

.

.

Python Standard Library: functools

Calculate 5!

1! = 1

2! = 1x2

3! = 1x2x3

4! = 1x2x3x4

.

.

Python Standard Library: functools

- Higher-order functions and operations
- Returns a function or takes another function as an argument
  - **reduce():** reducing iterable to a single cumulative value
  - **lru_cache():** memoizing callable and returns the stored value
  - **partial():** returns partial function which “freezes” some arguments

Python Standard Library: functools

```
def function(arg1, arg2, arg3):

        .

        .

        .
```

Python Standard Library: functools

def function(**arg1, arg2**, arg3):

.

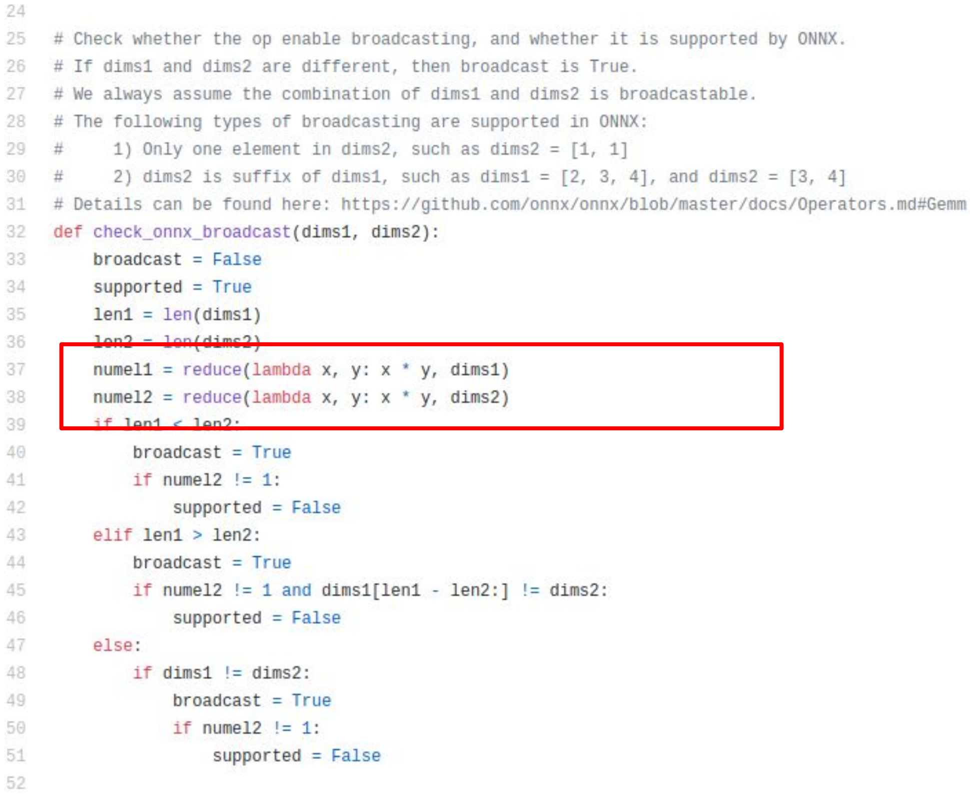
.

.

Notebook

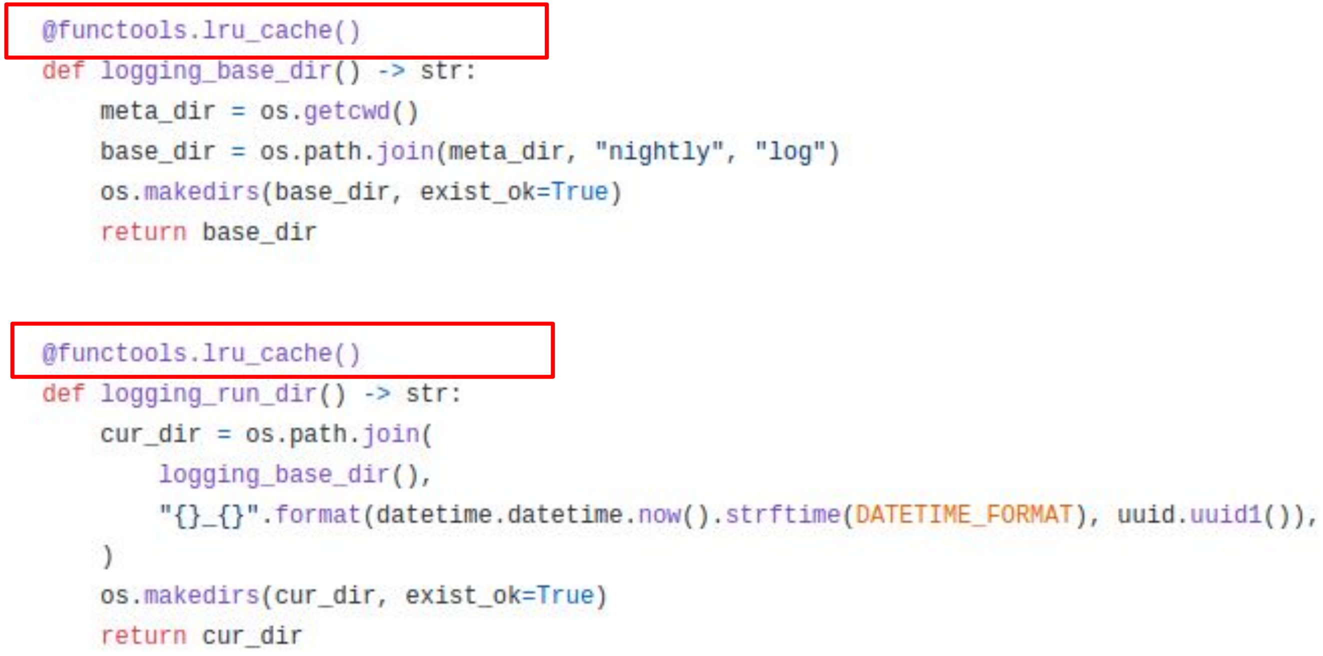
Applications: functools

Broadcasting Function in PyTorch Library



```
# Check whether the op enable broadcasting, and whether it is supported by ONNX.
# If dims1 and dims2 are different, then broadcast is True.
# We always assume the combination of dims1 and dims2 is broadcastable.
# The following types of broadcasting are supported in ONNX:
#     1) Only one element in dims2, such as dims2 = [1, 1]
#     2) dims2 is suffix of dims1, such as dims1 = [2, 3, 4], and dims2 = [3, 4]
# Details can be found here: https://github.com/onnx/onnx/blob/master/docs/Operators.md#Gemm
def check_onnx_broadcast(dims1, dims2):
    broadcast = False
    supported = True
    len1 = len(dims1)
    len2 = len(dims2)
    numel1 = reduce(lambda x, y: x * y, dims1)
    numel2 = reduce(lambda x, y: x * y, dims2)
    if len1 < len2:
        broadcast = True
        if numel2 != 1:
            supported = False
    elif len1 > len2:
        broadcast = True
        if numel2 != 1 and dims1[len1 - len2:] != dims2:
            supported = False
    else:
        if dims1 != dims2:
            broadcast = True
            if numel2 != 1:
                supported = False

```

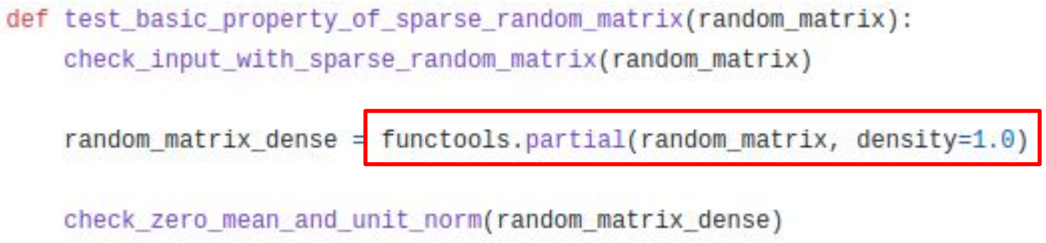
Applications: functools



```
@functools.lru_cache()
def logging_base_dir() -> str:
    meta_dir = os.getcwd()
    base_dir = os.path.join(meta_dir, "nightly", "log")
    os.makedirs(base_dir, exist_ok=True)
    return base_dir


@functools.lru_cache()
def logging_run_dir() -> str:
    cur_dir = os.path.join(
        logging_base_dir(),
        "{}_{}".format(datetime.datetime.now().strftime(DATETIME_FORMAT), uuid.uuid1()),
    )
    os.makedirs(cur_dir, exist_ok=True)
    return cur_dir
```

Applications: functools

Function from Scikit Library



```python
def test_basic_property_of_sparse_random_matrix(random_matrix):
    check_input_with_sparse_random_matrix(random_matrix)

    random_matrix_dense = functools.partial(random_matrix, density=1.0)

    check_zero_mean_and_unit_norm(random_matrix_dense)
```

Thank You

Standard Libraries: collections

Python Standard Libraries

In this module we will cover:

- Itertools
- functools
- collections
- pickle

Complete List: The Python Standard Library

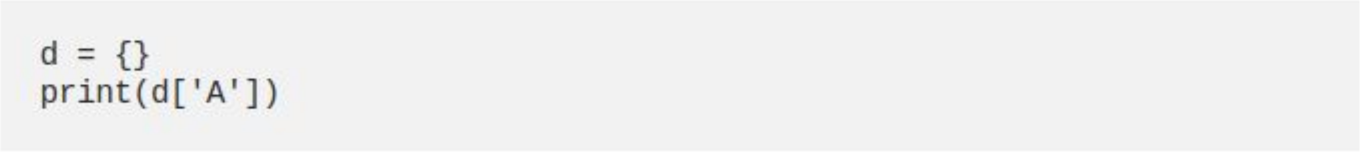
Python Standard Library: collections

- Specialized container datatypes providing alternatives to Python’s general purpose built-in containers, dict, list, set, and tuple.
- Alternative to dict, list, set, and tuple

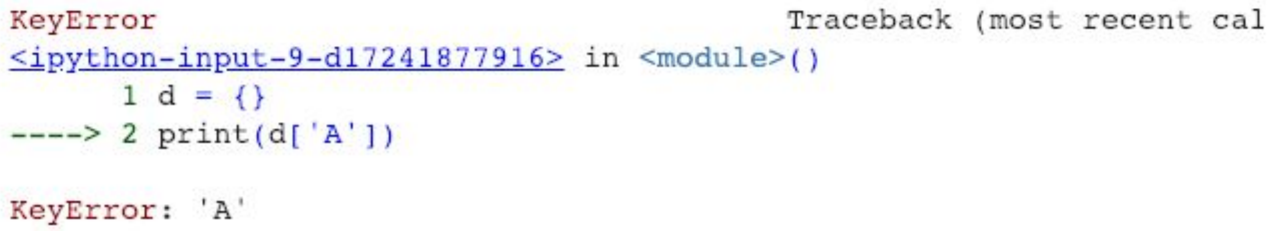
Python Standard Library: collections

- Specialized container datatypes providing alternatives to Python’s general purpose built-in containers, dict, list, set, and tuple.
- Alternative to dict, list, set, and tuple
    - **defaultdict():** provides default value for keys that do not exist

Python Standard Library: collections



```
d = {}
print(d['A'])
```



```
KeyError                                  Traceback (most recent cal
<ipython-input-9-d17241877916> in <module>()
      1 d = {}
----> 2 print(d['A'])

KeyError: 'A'
```

Python Standard Library: collections

- Specialized container datatypes providing alternatives to Python’s general purpose built-in containers, dict, list, set, and tuple.
- Alternative to dict, list, set, and tuple
    - **defaultdict():** provides default value for keys that do not exist
    - **counter():** dict subclass which helps to count hashable objects

Python Standard Library: collections

text = ‘Analytics Vidhya is a platform to learn about Data Science,
Machine Learning, Deep Learning, Data Visualisations,
Business Analytics, Big Data and more’

Count of words in text:

Analytics: 2

Vidhya: 1

.

.

Data: 3

.

.

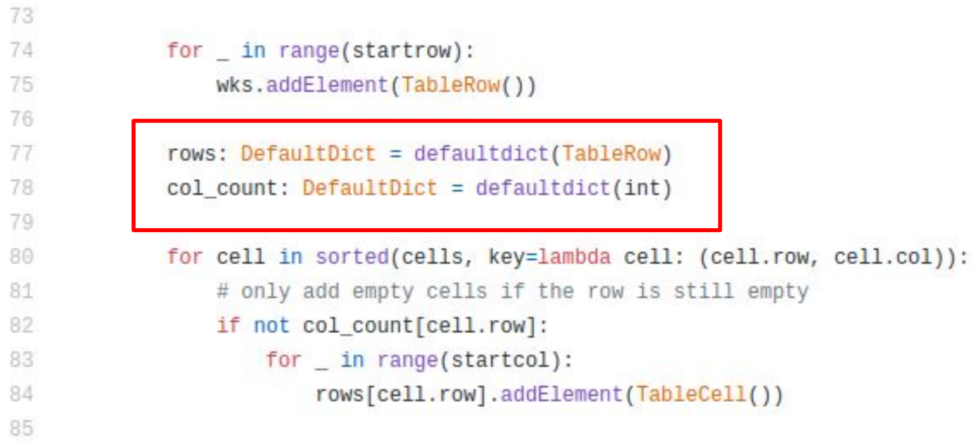
Python Standard Library: collections

- Specialized container datatypes providing alternatives to Python’s general purpose built-in containers, dict, list, set, and tuple.
- Alternative to dict, list, set, and tuple
  - **defaultdict():** provides default value for keys that do not exist
  - **counter():** dict subclass which helps to count hashable objects
  - **deque():** double ended queue for fast appends and pops from either end

Notebook

Applications: collections

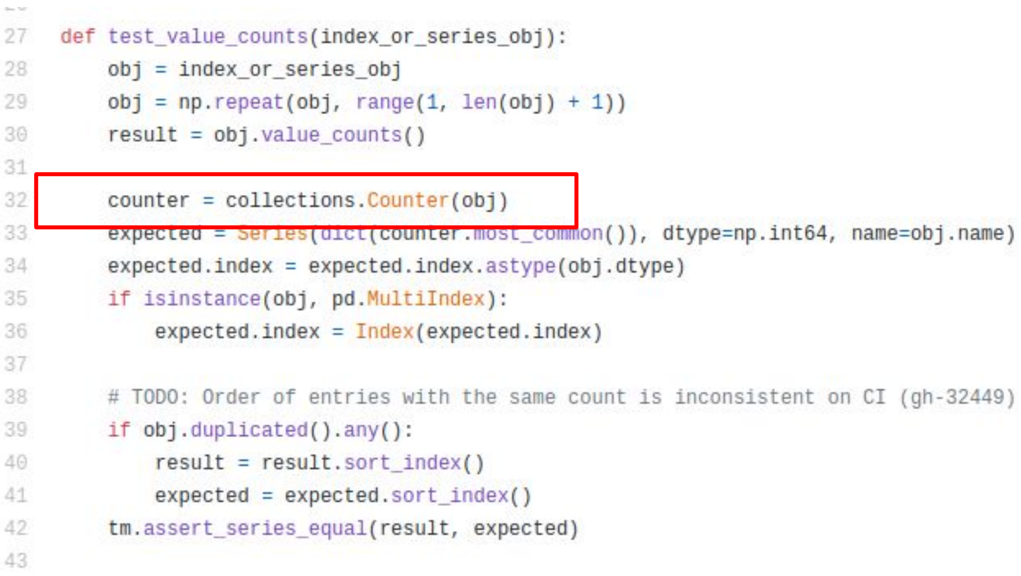
Pandas Implementation



```
        for _ in range(startrow):
            wks.addElement(TableRow())

        rows: DefaultDict = defaultdict(TableRow)
        col_count: DefaultDict = defaultdict(int)

        for cell in sorted(cells, key=lambda cell: (cell.row, cell.col)):
            # only add empty cells if the row is still empty
            if not col_count[cell.row]:
                for _ in range(startcol):
                    rows[cell.row].addElement(TableCell())

```

Applications: collections

Pandas Implementation



```python
def test_value_counts(index_or_series_obj):
    obj = index_or_series_obj
    obj = np.repeat(obj, range(1, len(obj) + 1))
    result = obj.value_counts()

    counter = collections.Counter(obj)
    expected = Series(dict(counter.most_common()), dtype=np.int64, name=obj.name)
    expected.index = expected.index.astype(obj.dtype)
    if isinstance(obj, pd.MultiIndex):
        expected.index = Index(expected.index)

    # TODO: Order of entries with the same count is inconsistent on CI (gh-32449)
    if obj.duplicated().any():
        result = result.sort_index()
        expected = expected.sort_index()
    tm.assert_series_equal(result, expected)

```

Thank You

Standard Libraries: pickle

Analytics Vidhya

Python Standard Libraries

In this module we will cover:

- Itertools
- functools
- collections
- pickle

Complete List: The Python Standard Library

Python Standard Library: pickle

- Used for serializing and de-serializing a Python object structure

Python Standard Library: pickle

- Used for serializing and de-serializing a Python object structure
- **Serializing:** The process to converts any kind of python objects (list, dict, etc.) into byte streams (0s and 1s)
- **De-serializing:** converts the byte stream (generated through pickling) back into python objects

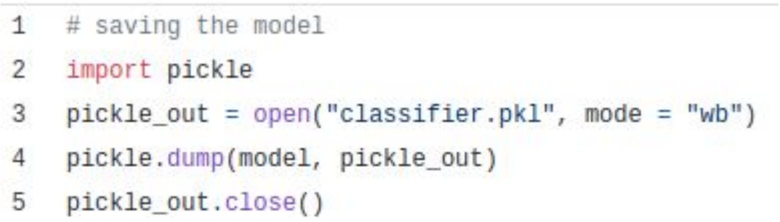
Python Standard Library: pickle

- Used for serializing and de-serializing a Python object structure
- **Serializing:** The process to converts any kind of python objects (list, dict, etc.) into byte streams (0s and 1s)
- **De-serializing:** converts the byte stream (generated through pickling) back into python objects
- dump() and load() are the functions used for pickling and unpickling

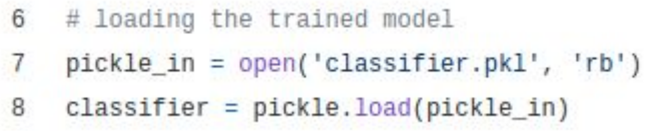
Notebook

Applications of Standard Library: pickle

Deploying machine learning models using Streamlit



```python
# saving the model
import pickle
pickle_out = open("classifier.pkl", mode = "wb")
pickle.dump(model, pickle_out)
pickle_out.close()
```



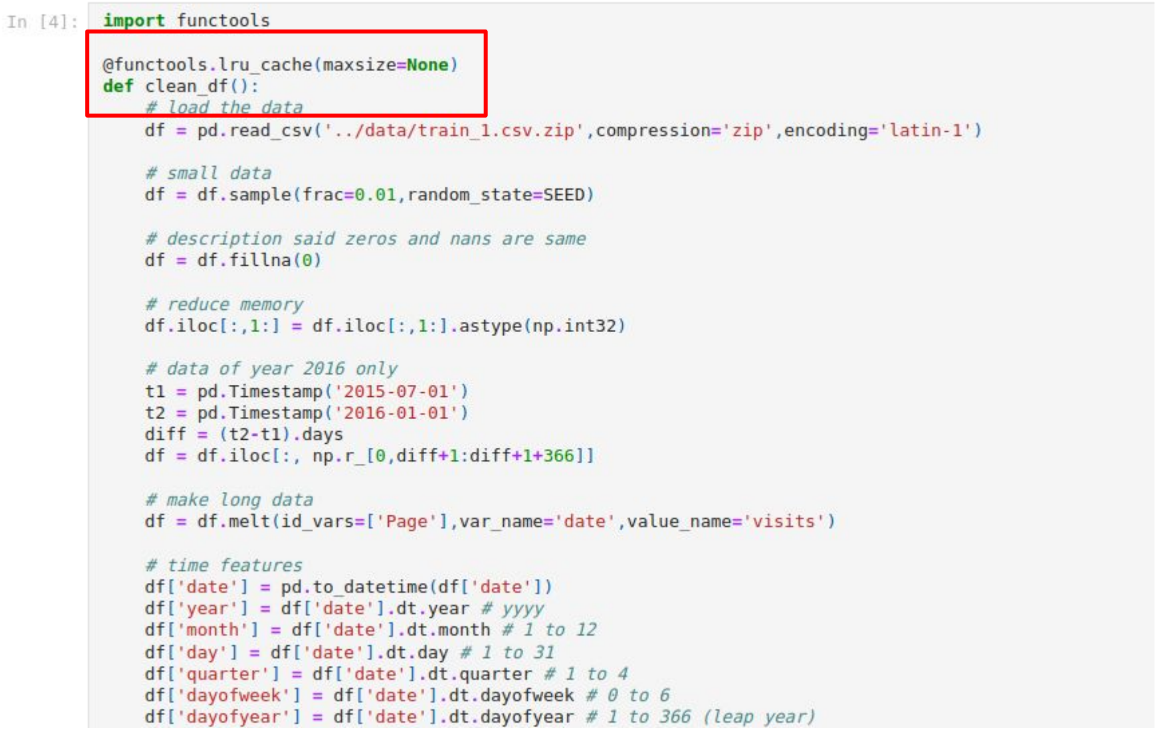
```python
# loading the trained model
pickle_in = open('classifier.pkl', 'rb')
classifier = pickle.load(pickle_in)
```

Thank You

Analytics Vidhya

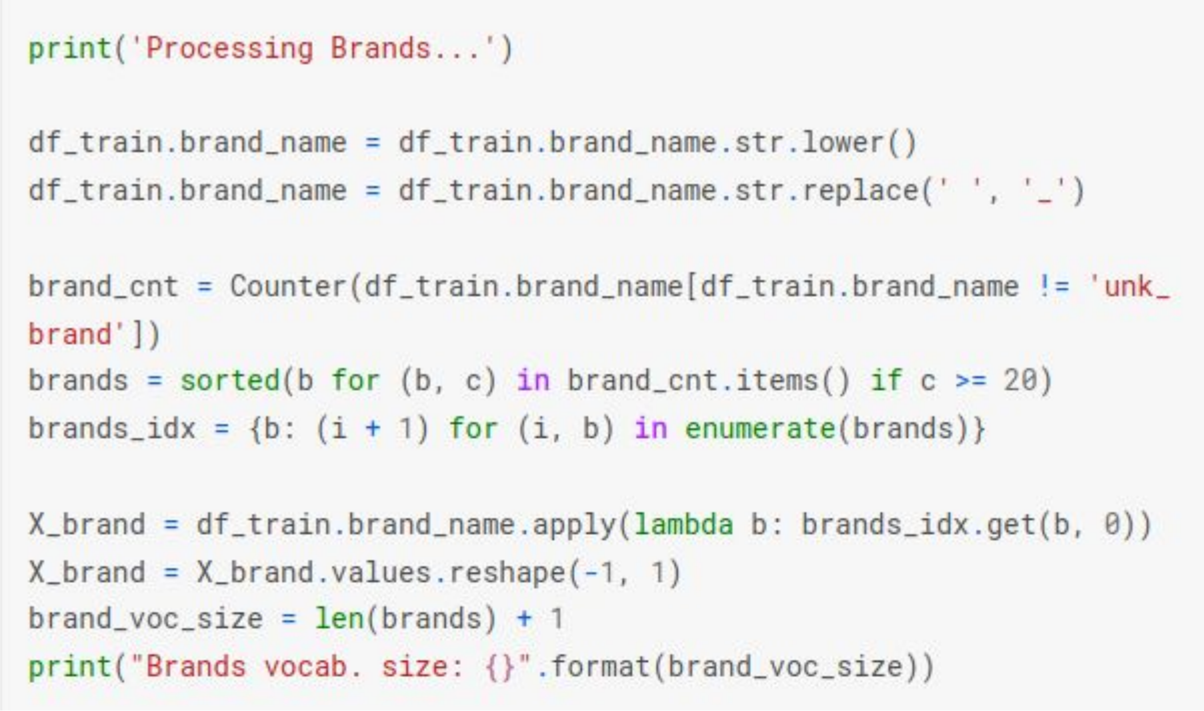
Applications of Standard Library: functools

Web Traffic Forecasting Problem



```python
import functools

@functools.lru_cache(maxsize=None)
def clean_df():
    # load the data
    df = pd.read_csv('../data/train_1.csv.zip',compression='zip',encoding='latin-1')

    # small data
    df = df.sample(frac=0.01,random_state=SEED)

    # description said zeros and nans are same
    df = df.fillna(0)

    # reduce memory
    df.iloc[:,1:] = df.iloc[:,1:].astype(np.int32)

    # data of year 2016 only
    t1 = pd.Timestamp('2015-07-01')
    t2 = pd.Timestamp('2016-01-01')
    diff = (t2-t1).days
    df = df.iloc[:, np.r_[0,diff+1:diff+1+366]]

    # make long data
    df = df.melt(id_vars=['Page'],var_name='date',value_name='visits')

    # time features
    df['date'] = pd.to_datetime(df['date'])
    df['year'] = df['date'].dt.year # yyyy
    df['month'] = df['date'].dt.month # 1 to 12
    df['day'] = df['date'].dt.day # 1 to 31
    df['quarter'] = df['date'].dt.quarter # 1 to 4
    df['dayofweek'] = df['date'].dt.dayofweek # 0 to 6
    df['dayofyear'] = df['date'].dt.dayofyear # 1 to 366 (leap year)
```

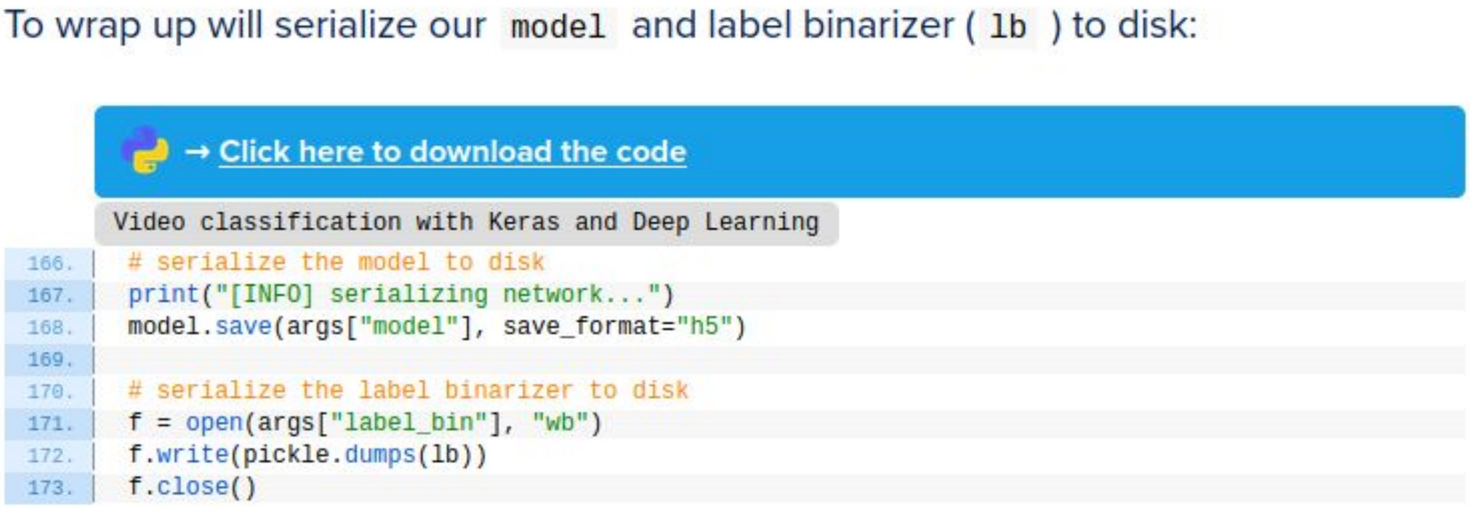
Applications of Standard Library: collections



```
print('Processing Brands...')

df_train.brand_name = df_train.brand_name.str.lower()
df_train.brand_name = df_train.brand_name.str.replace(' ', '_')

brand_cnt = Counter(df_train.brand_name[df_train.brand_name != 'unk_
brand'])
brands = sorted(b for (b, c) in brand_cnt.items() if c >= 20)
brands_idx = {b: (i + 1) for (i, b) in enumerate(brands)}

X_brand = df_train.brand_name.apply(lambda b: brands_idx.get(b, 0))
X_brand = X_brand.values.reshape(-1, 1)
brand_voc_size = len(brands) + 1
print("Brands vocab. size: {}".format(brand_voc_size))
```

Applications of Standard Library: pickle

Video classification with Keras and Deep Learning



To wrap up will serialize our `model` and label binarizer (`lb`) to disk:

Video classification with Keras and Deep Learning

```python
# serialize the model to disk
print("[INFO] serializing network...")
model.save(args["model"], save_format="h5")

# serialize the label binarizer to disk
f = open(args["label_bin"], "wb")
f.write(pickle.dumps(lb))
f.close()
```