Python DataTypes

FPT Education
FPT UNIVERSITY

Series and List

Array and Matrix

DataFrame

Problem Solving: BigMartSales Prediction

List

- The **list is a type of data** in Python used to **store multiple objects**. It is an **ordered and mutable collection** of comma-separated items between square brackets
- List constants are surrounded by square brackets and the elements in the list are separated by commas
- A list element can be any Python object - even another list

friends ['Joseph', 'Glenn', 'Sally']

carryon ['socks', 'shirt', 'perfume']

```
>>> print([1, 24, 76])
[1, 24, 76]
>>> print(['red', 'yellow', 'blue’])
['red', 'yellow', 'blue']
```

- The len() function takes a list as a parameter and returns the number of elements in the list

```
>>> greet = 'Hello Bob'
>>> print(len(greet))
>>> x = [ 1, 2, 'joe', 99]
>>> print(len(x))
```

- The range function returns a list of numbers that range from zero to one less than the parameter

```
>>> print(range(4))
[0, 1, 2, 3]
>>> friends = ['Joseph', 'Glenn', 'Sally']
>>> print(len(friends))
3
>>> print(list(range(len(friends))))
[0, 1, 2]
```

List

- Lists Can Be Sliced Using
- Concatenating Lists Using +
- List Methods: index(), count(), sort(), insert(), append(), remove(), pop()

```
>>> x = list()
>>> type(x)
<type 'list'>
>>> dir(x)
[... 'append', 'count', 'extend',
'index', 'insert', 'pop', 'remove',
'reverse', 'sort']
>>>
```

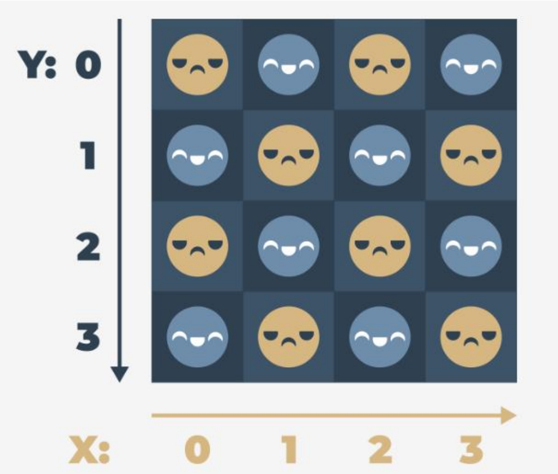
```
>>> t = [9, 41, 12, 3, 74, 15]
>>> t[1:3]
[41,12]
>>> t[:4]
[9, 41, 12, 3]
>>> t[3:]
[3, 74, 15]
>>> t[:]
[9, 41, 12, 3, 74, 15]
```

```
>>> a = [1, 2, 3]
>>> b = [4, 5, 6]
>>> c = a + b
>>> print(c)
[1, 2, 3, 4, 5, 6]
>>> print(a)
[1, 2, 3]
```

List

- You can use nested lists in Python to create matrices (i.e., two-dimensional lists). For example:
- You can nest as many lists in lists as you want, thereby creating n-dimensional lists, e.g., three-, four- or even sixty-four-dimensional arrays. For example:

```
table = [[":(", ":)", ":(", ":)"],
         [":)", ":(", ":)", ":)"],
         [":(", ":)", ":)", ":("],
         [":)", ":)", ":)", ":("]]

print(table)
print(table[0][0])  # outputs: ':('
print(table[0][3])  # outputs: ':)'
```



Accessing list content

- Get single element in a list using an index specified in square brackets

```
my_list = [1, 2, 3, 4]
del my_list[2]
print(my_list)   # outputs: [1, 2, 4]

del my_list  # deletes the whole list
```

```
>>> friends = [ 'Joseph',
'Glenn', 'Sally' ]
>>> print(friends[1])
Glenn
>>>
```

- Lists can be **iterated** through using the `for` loop, e.g.:

```
my_list = ["white", "purple", "blue", "yellow", "green"]

for color in my_list:
    print(color)
```

Sequence types and mutability

- A sequence type is a type of data in Python which is able to store more than one value (or less than one, as a sequence may be empty), and these values can be sequentially (hence the name) browsed, element by element.
  - As the for loop is a tool especially designed to iterate through sequences, we can express the definition as: a sequence is data which can be scanned by the for loop.
- The second notion - mutability - is a property of any of Python's data that describes its readiness to be freely changed during program execution. There are two kinds of Python data: mutable and immutable.
  - Mutable data can be freely updated at any time - we call such an operation in situ.
  - Immutable data cannot be modified in this way.

What is a tuple?

- The first and the clearest distinction between lists and tuples is the syntax used to create them - tuples prefer to use parenthesis, whereas lists like to see brackets, although it's also possible to create a tuple just from a set of values separated by commas.
- each tuple element may be of a different type (floating-point, integer, or any other not-as-yet-introduced kind of data).
  - the len() the
  - + operator
  - the * operator
  - the in and not in operators

```
my_tuple = (1, 10, 100)
t1 = my_tuple + (1000, 10000)
t2 = my_tuple * 3
print(len(t2))
print(t1)
print(t2)
print(10 in my_tuple)
print(-10 not in my_tuple)
```

Sets

- A set is a unique collection of objects in Python.
- You can denote a set with a curly bracket {}.
- Python will automatically remove duplicate items:
  - set1 = {"pop", "rock", "soul", "hard rock", "rock", "R&B", "rock", "disco"}
- Convert list to set: set()
- Set operations:
  - add(), remove() method
  - intersection = set1&set2, set1.intersection(set2), set2.intersection(set1)
  - set1.union(set2)
- Check if subset, superset: set(set1).issuperset(set2)

What is a dictionary?

- The dictionary is another Python data structure. It's not a sequence type (but can be easily adapted to sequence processing) and it is mutable.
- The Python dictionary works in the same way as a bilingual dictionary
- This means that a dictionary is a set of key-value pairs. Note:
  - each key must be unique - it's not possible to have more than one key of the same value;
  - a key may be any immutable type of object: it can be a number (integer or float), or even a string, but not a list;
  - a dictionary is not a list - a list contains a set of numbered values, while a dictionary holds pairs of values;
  - the len() function works for dictionaries, too - it returns the numbers of key-value elements in the dictionary;
  - a dictionary is a one-way tool - if you have an English-French dictionary, you can look for French equivalents of English terms, but not vice versa.

How to make a dictionary?

- use the following syntax:

```
dictionary = {"cat": "chat", "dog": "chien", "horse": "cheval"}
phone_numbers = {'boss': 5551234567, 'Suzy': 22657854310}
empty_dictionary = {}

print(dictionary)
print(phone_numbers)
print(empty_dictionary)
```

- The list of pairs is surrounded by curly braces, while the pairs themselves are separated by commas, and the keys and values by colons.
- If you want to get any of the values, you have to deliver a valid key value

The items() and values() methods

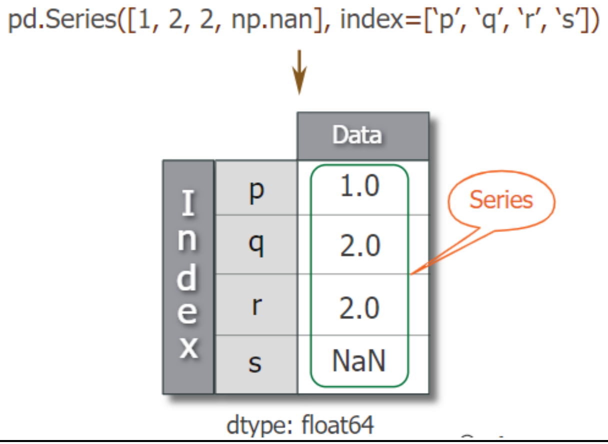
- items(). The method returns tuples (this is the first example where tuples are something more than just an example of themselves) where each tuple is a key-value pair.
  - values(), which works similarly to keys(), but returns values.
  - modifying and adding values
  - update(), popitem()
  - you can use the for loop in dictionary

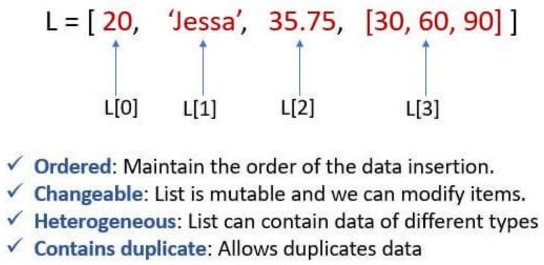
```
dictionary = {"cat": "chat", "dog": "chien", "horse": "cheval"}

for english, french in dictionary.items():
    print(english, "->", french)
```

```
dictionary = {"cat": "chat", "dog": "chien", "horse": "cheval"}

dictionary['cat'] = 'minou'
print(dictionary)
```

Series

- **Series**: *Series is a one-dimensional labeled array capable of holding data of any type (integer, string, float, python objects, etc.). The axis labels are collectively called index*
- ***<u>Example</u>***: *x = pd.Series([1, 2, 2, np.nan], index=['p', 'q', 'r', 's'])*





Array and Matrix

- **Array**: *Array's are a data structure for storing homogeneous data. That mean's all elements are the same type.*
- ***Example***:

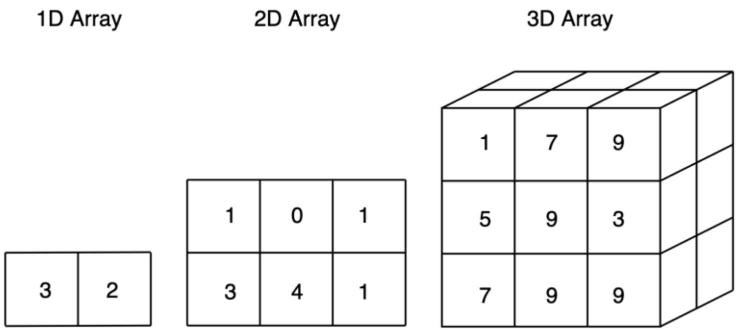
  arr = np.array([[1,2],[3,4]])

- ***Matrix***: *A matrix is a two-dimensional rectangular array in which data are arranged in the form of rows and columns. The horizontal entries in a matrix are called rows and the vertical entries in a matrix are called columns.*
- ***Example***:

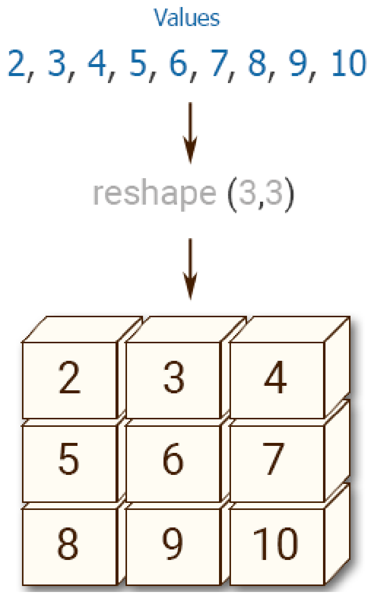
  mat = [[1, 2, 3], [6, 5, 4]]

Array and Matrix

Array



Matrix

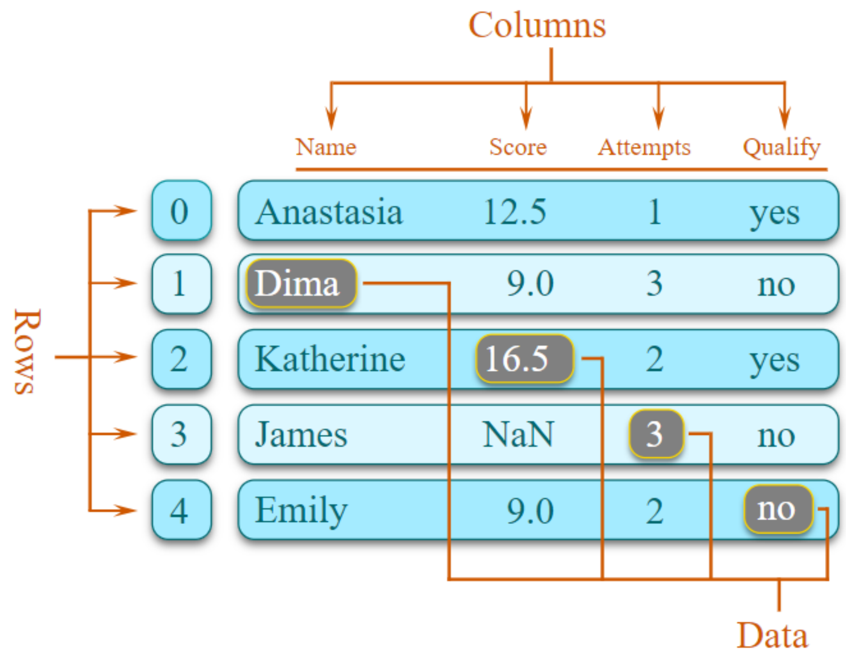


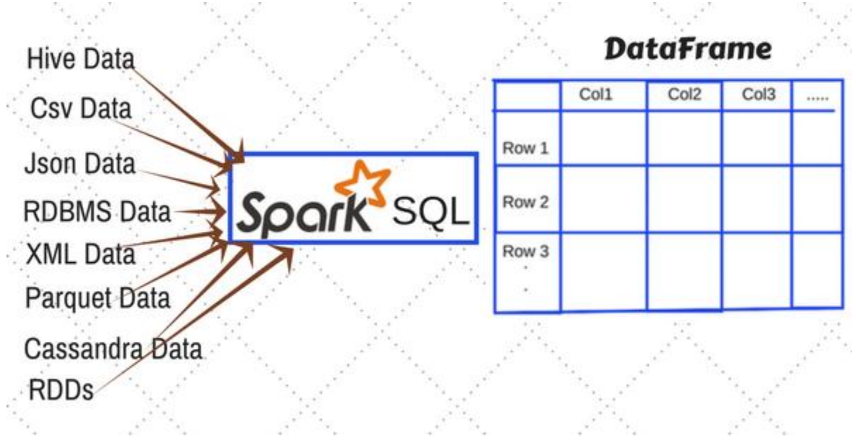
DataFrame

- **DataFrame**: *DataFrame is two-dimensional size-mutable, potentially heterogeneous tabular data structure with labeled axes (rows and columns). A Data frame is a two-dimensional data structure, i.e., data is aligned in a tabular fashion in rows and columns. Pandas DataFrame consists of three principal components, the data, rows, and columns. .*
- ***Example***:

```
df = pd.DataFrame({ 'X':[78,85,96,80,86],
                    'Y':[84,94,89,83,86],
                    'Z':[86,97,96,72,83]});
```

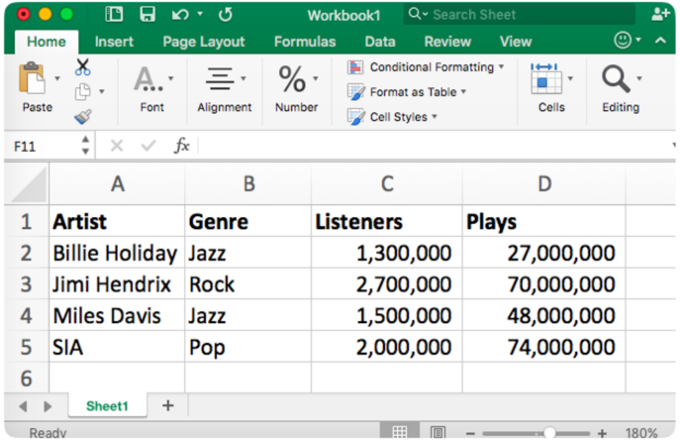
DataFrame


Columns
Name
Score
Attempts
Qualify
Rows
0
Anastasia
12.5
1
yes
1
Dima
9.0
3
no
2
Katherine
16.5
2
yes
3
James
NaN
3
no
4
Emily
9.0
2
no
Data


Hive Data
Csv Data
Json Data
RDBMS Data
XML Data
Parquet Data
Cassandra Data
RDDs
Spark SQL
DataFrame
Col1
Col2
Col3
.....
Row 1
Row 2
Row 3

DataFrame

`music.csv`



	A	B	C	D
1	**Artist**	**Genre**	**Listeners**	**Plays**
2	Billie Holiday	Jazz	1,300,000	27,000,000
3	Jimi Hendrix	Rock	2,700,000	70,000,000
4	Miles Davis	Jazz	1,500,000	48,000,000
5	SIA	Pop	2,000,000	74,000,000
6				

`pandas.read_csv('music.csv')`

	Artist	Genre	Listeners	Plays
0	Billie Holiday	Jazz	1,300,000	27,000,000
1	Jimi Hendrix	Rock	2,700,000	70,000,000
2	Miles Davis	Jazz	1,500,000	48,000,000
3	SIA	Pop	2,000,000	74,000,000

DataFrame

- Viewing/Inspecting Data

Function	Description
df.head(n)	First n rows of the DataFrame
df.tail(n)	Last n rows of the DataFrame
df.shape	Number of rows and columns
df.info()	Index, Datatype and Memory information
df.describe()	Summary statistics for numerical columns
s.value_counts(dropna=False)	View unique values and counts
df.apply(pd.Series.value_counts)	Unique values and counts for all columns

DataFrame

- Selection

Function	Description
df[col]	Returns column with label col as Series
df[[col1, col2]]	Returns columns as a new DataFrame
s.iloc[0]	Selection by position
s.loc['index_one']	Selection by index
df.iloc[0,:]	First row
df.iloc[0,0]	First element of first column

DataFrame

- Data Cleaning

Function	Description
df.columns = ['a','b','c']	Rename columns
pd.isnull()	Checks for null Values, Returns Boolean Arrray
pd.notnull()	Opposite of pd.isnull()
df.dropna()	Drop all rows that contain null values
df.dropna(axis=1)	Drop all columns that contain null values
df.dropna(axis=1,thresh=n)	Drop all rows have less than n non null values
df.fillna(x)	Replace all null values with x

DataFrame

Function (Data Cleaning)	Description
s.fillna(s.mean())	Replace all null values with the mean
s.astype(float)	Convert the datatype of series to float
s.replace(1,'one')	Replace all values equal to 1 with 'one'
s.replace([2,3],['two', 'three'])	Replace all 2 with 'two' 3 with 'three'
df.rename(columns=lambda x: x + 1)	Mass renaming of columns
df.rename(columns={'old_name': 'new_ name'}) : Selective renaming	
df.set_index('column_one')	Change the index
df.rename(index=lambda x: x + 1)	Mass renaming of index

DataFrame

Filter, Sort, and Groupby

Function	Description
df[df[col] > 0.6]	Rows where the column col is greater than 0.6
df[(df[col] > 0.6) & (df[col] < 0.8)]	Rows where 0.8 > col > 0.6
df.sort_values(col1)	Sort values by col1 in ascending order
df.sort_values(col2,ascending=False)	Sort values by col2 in descending order.5

FPT Education
FPT UNIVERSITY

Filter, Sort, and Groupby

Function	Description
df.sort_values([col1,col2],ascending =[True,False])	Sort values by col1 in ascending order then col2 in descending order
df.apply(np.mean)	Apply the function np.mean() across each column
df.apply(np.max,axis=1)	Apply the function np.max() across each row

- Filter, Sort, and Groupby

Function	Description
df.groupby(col)	Returns a groupby object for values from one column
df.groupby([col1,col2])	Returns groupby object for values from multiple columns
df.groupby(col1)[col2]	Returns the mean of the values in col2, grouped by the values in col1

DataFrame

- Filter, Sort, and Groupby

Function	Description
df.pivot_table(index=col1,values=[col2,col3],aggfunc=mean)	Create a pivot table that groups by col1 and calculates the mean of col2 and col3
df.groupby(col1).agg(np.mean)	Find the average across all columns for every unique col1 group
df.rename(index=lambda x: x + 1)	Mass renaming of index

DataFrame

- Join/Combine

Function	Description
df1.append(df2)	Add the rows in df1 to the end of df2 (columns should be identical)
pd.concat([df1, df2],axis=1)	Add the columns in df1 to the end of df2 (rows should be identical)
df1.join(df2,on=col1, how='inner')	SQL-style join the columns in df1 with the columns on df2 where the rows for col have identical values. The 'how' can be 'left', 'right', 'outer' or 'inner'

DataFrame

- Statistics

Function	Description
df.describe()	Summary statistics for numerical columns
df.mean()	Returns the mean of all columns
df.corr()	Returns the correlation between columns in a DataFrame
df.count()	Returns the number of non-null values in each DataFrame column

DataFrame

- Statistics

Function	Description
df.max()	Returns the highest value in each column
df.min()	Returns the lowest value in each column
df.median()	Returns the median of each column
df.std()	Returns the standard deviation of each column

DataFrame

- Importing Data

Function	Description
pd.read_csv(filename)	From a CSV file
pd.read_table(filename)	From a delimited text file (like TSV)
pd.read_excel(filename)	From an Excel file
pd.read_sql(query, connection_object)	Read from a SQL table/database

DataFrame

- Importing Data

Function	Description
pd.read_json(json_string)	Read from a JSON formatted string, URL or file.
pd.read_html(url)	Parses an html URL, string or file and extracts tables to a list of dataframes
pd.read_clipboard()	Takes the contents of your clipboard and passes it to read_table()
pd.DataFrame(dict)	From a dict, keys for columns names, values for data as lists

DataFrame

- Exporting Data

Function	Description
df.to_csv(filename)	Write to a CSV file
df.to_excel(filename)	Write to an Excel file
df.to_sql(table_name, connection_object)	Write to a SQL table
df.to_json(filename)	Write to a file in JSON format

Problem Solving

BigMartSales Prediction

- The data scientists at BigMart have collected 2013 sales data for 1559 products across 10 stores in different cities. Also, certain attributes of each product and store have been defined.
- The aim is to build a predictive model and find out the sales of each product at a particular store. Using this model, BigMart will try to understand the properties of products and stores which play a key role in increasing sales

FPT Education
FPT UNIVERSITY

BigMartSales Prediction

Variable	Description	Relation to Hypothesis
Item_Identifier	Unique product ID	ID Variable
Item_Weight	Weight of product	Not considered in hypothesis
Item_Fat_Content	Whether the product is low fat or not	Linked to 'Utility' hypothesis. Low fat items are generally used more than others
Item_Visibility	The % of total display area of all products in a store allocated to the particular product	Linked to 'Display Area' hypothesis.
Item_Type	The category to which the product belongs	More inferences about 'Utility' can be derived from this.
Item_MRP	Maximum Retail Price (list price) of the product	Not considered in hypothesis
Outlet_Identifier	Unique store ID	ID Variable
Outlet_Establishment_Year	The year in which store was established	Not considered in hypothesis
Outlet_Size	The size of the store in terms of ground area covered	Linked to 'Store Capacity' hypothesis
Outlet_Location_Type	The type of city in which the store is located	Linked to 'City Type' hypothesis.
Outlet_Type	Whether the outlet is just a grocery store or some sort of supermarket	Linked to 'Store Capacity' hypothesis again.
Item_Outlet_Sales	Sales of the product in the particular store. This is the outcome variable to be predicted.	Outcome variable