**The Ultimate Guide to the Pandas Library for Data Science in Python**

The package is known for a very useful data structure called the pandas DataFrame. Pandas also allows Python developers to easily deal with tabular data (like spreadsheets) within a Python script.

*[pandas] is derived from the term "panel data", an econometrics term for data sets that include observations over multiple time periods for the same individuals. — Wikipedia*

### Article Resources:

## What's Pandas for?

This tool is essentially your data’s home. Through pandas, you get acquainted with your data by cleaning, transforming, and analyzing it

1-Load & read the data

2-Cleaning the data

3-Transforming the data to meaningful information.

4- Calculate statistics

5- Clean the data by doing things like removing missing values and filtering rows or columns by some criteria

6- Visualize the data with help from Matplotlib. Plot bars, lines, histograms, bubbles, and more.

7- Store the cleaned, transformed data back into a CSV, other file or database

Note: You should have solid understanding of data

**Shall start for pandas:**

## Pandas First Steps:

### Install and import

Pandas is an easy package to install. Open up your terminal program (for Mac users) or command line (for PC users) and install it using either of the following commands:

pip install pandas(Restart the kernel)

import pandas as pd

The primary two components of pandas are the Series and DataFrame.

A Series is essentially a column, and a DataFrame is a multi-dimensional table made up of a collection of Series.

### Creating DataFrames from scratch

There are *many* ways to create a DataFrame from scratch, but a great option is to just use a simple dict.

data = {

'seq\_1': [3, 2, 0, 1],

'seq\_2': [0, 3, 7, 2]

}

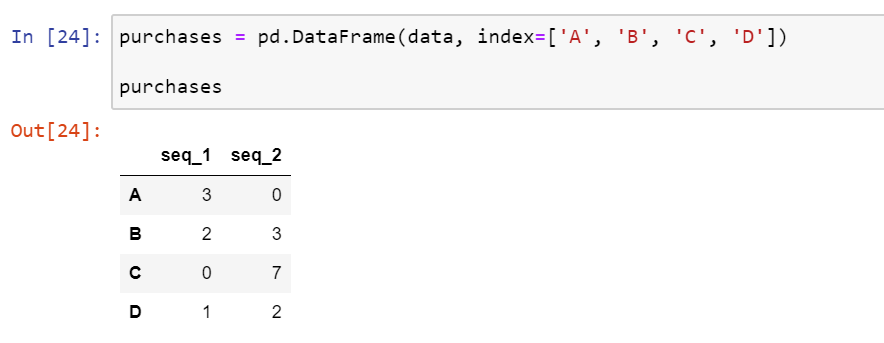
And then pass it to the pandas DataFrame constructor:

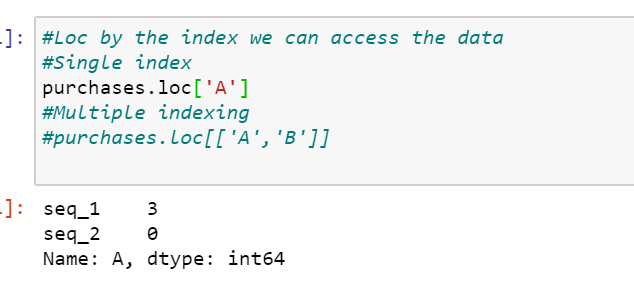
**Python** | **Pandas DataFrame**. **Pandas DataFrame** is two-dimensional size-mutable, potentially heterogeneous tabular data structure with labeled axes (rows and columns). A **Data frame** is a two-dimensional data structure, i.e., data is aligned in a tabular fashion in rows and columns.

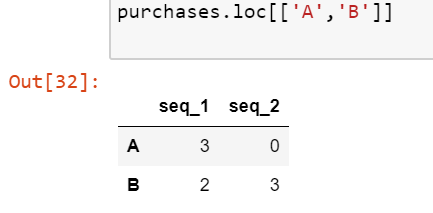
purchases = pd.DataFrame(data)



Changing the index to A B C D







#### Splitting Data into Groups

There are multiple ways to split data like:

* obj.groupby(key)
* obj.groupby(key, axis=1)
* obj.groupby([key1, key2])

Note that groupby will preserve the order in which observations are sorted within each group. For example, the groups created by groupby() below are in the order they appeared in the original DataFrame

**In [24]:** df3 = pd.DataFrame({'X': ['A', 'B', 'A', 'B'], 'Y': [1, 4, 3, 2]})

**In [25]:** df3.groupby(['X']).get\_group('A')

**Out[25]:**

X Y

0 A 1

2 A 3

**In [26]:** df3.groupby(['X']).get\_group('B')

Out[26]:

X Y

1 B 4

3 B 2

**Dataframe on where:**

Pandas **where()** method is used to check a data frame for one or more condition and return the result accordingly. By default, The rows not satisfying the condition are filled with NaN value.

***Syntax:******DataFrame.where****(cond, other=nan, inplace=False, axis=None, level=None, errors=’raise’, try\_cast=False, raise\_on\_error=None)*

***Parameters:***

***cond:****One or more condition to check data frame for.****other:****Replace rows which don’t satisfy the condition with user defined object, Default is NaN****inplace:****Boolean value, Makes changes in data frame itself if True****axis:****axis to check( row or columns)*

|  |
| --- |
| # importing pandas package  import pandas as pd    # making data frame from csv file  data = pd.read\_csv("nba.csv")    # sorting dataframe  data.sort\_values("Team", inplace = True)    # making boolean series for a team name  filter1 = data["Team"]=="Atlanta Hawks"    # making boolean series for age  filter2 = data["Age"]>24    # filtering data on basis of both filters  data.where(filter1 & filter2, inplace = True)    # display  data |

## 1. Pandas DataFrame dropna() Function

Pandas DataFrame dropna() function is used to remove rows and columns with Null/NaN values. By default, this function returns a new DataFrame and the source DataFrame remains unchanged.

We can create null values using None, pandas.NaT, and numpy.nan variables.

The dropna() function syntax is:

dropna(self, axis=0, how="any", thresh=None, subset=None, inplace=False)

* **axis**: possible values are {0 or ‘index’, 1 or ‘columns’}, default 0. If 0, drop rows with null values. If 1, drop columns with missing values.
* **how**: possible values are {‘any’, ‘all’}, default ‘any’. If ‘any’, drop the row/column if any of the values is null. If ‘all’, drop the row/column if all the values are missing.
* **thresh**: an int value to specify the threshold for the drop operation.
* **subset**: specifies the rows/columns to look for null values.
* **inplace**: a boolean value. If True, the source DataFrame is changed and None is returned.

# Python | Pandas Dataframe.sample()

Pandas **sample()** is used to generate a sample random row or column from the function caller data frame

***Syntax:***

*DataFrame.sample(n=None, frac=None, replace=False, weights=None, random\_state=None, axis=None)*

***Parameters***

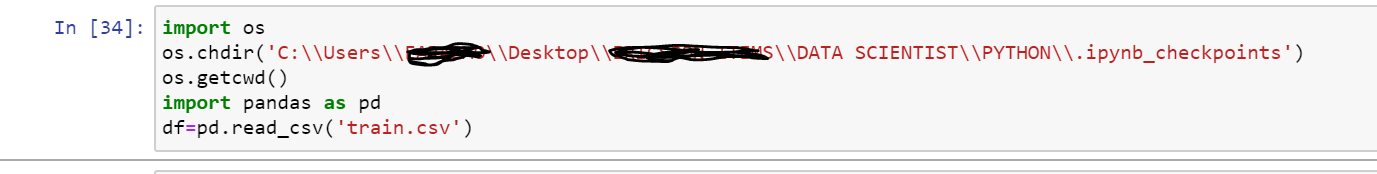
***n:****int value, Number of random rows to generate.****frac:****Float value, Returns (float value \* length of data frame values ). frac cannot be used with n.****replace:****Boolean value, return sample with replacement if True.****random\_state:****int value or numpy.random.RandomState, optional. if set to a particular integer, will return same rows as sample in every iteration.****axis:****0 or ‘row’ for Rows and 1 or ‘column’ for Columns.*

Reading different file format:

We have different kind of file format is there to read it.

>Make sure that Ur desired file should in that current directory otherwise file not found error will come.

Please follow the below steps to read a file without any error.

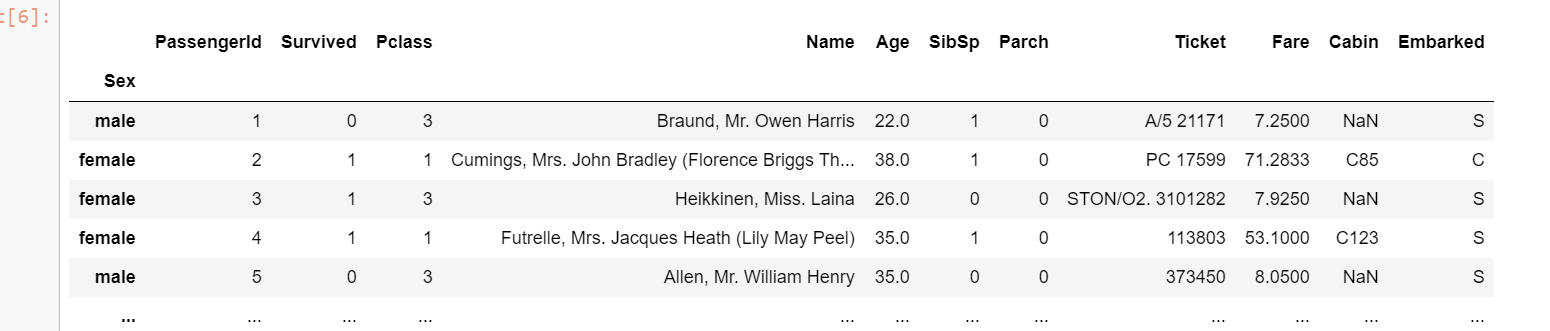


### Converting back to a CSV, JSON, or SQL

df.to\_csv('new\_purchases.csv')

df.to\_json('new\_purchases.json')





**import** **pandas** **as** **pd**

# Create some Pandas dataframes from some data.

df1 = pd.DataFrame({'Data': [11, 12, 13, 14]})

df2 = pd.DataFrame({'Data': [21, 22, 23, 24]})

df3 = pd.DataFrame({'Data': [31, 32, 33, 34]})

# Create a Pandas Excel writer using XlsxWriter as the engine.

writer = pd.ExcelWriter('pandas\_multiple.xlsx', engine='xlsxwriter')

# Write each dataframe to a different worksheet.

df1.to\_excel(writer, sheet\_name='Sheet1')

df2.to\_excel(writer, sheet\_name='Sheet2')

df3.to\_excel(writer, sheet\_name='Sheet3')

# Close the Pandas Excel writer and output the Excel file.

writer.save()

## Pandas DataFrame – Add or Insert Row

To append or add a row to DataFrame, create the new row as Series and use DataFrame.append() method.

In this tutorial, we shall learn how to append a row to an existing DataFrame, with the help of illustrative example programs.

### Syntax – append()

Following is the syntax of DataFrame.appen() function.

mydataframe = mydataframe.append(new\_row, ignore\_index=True)

where the resulting DataFrame contains **new\_row** added to **mydataframe**.

append() is immutable. It does not change the DataFrame, but returns a new DataFrame with the row appended.

### Example 1: Add Row to DataFrame

In this example, we will create a DataFrame and append a new row to this DataFrame. The new row is initialized as a Python Dictionary and append() function is used to append the row to the dataframe.

When you are adding a Python Dictionary to append(), make sure that you pass ignore\_index=True.

The append() method returns the dataframe with the newly added row.

**Python Program**

import pandas as

import pandas as pd

data = {'name': ['Somu', 'Kiku', 'Amol', 'Lini'],

'physics': [68, 74, 77, 78],

'chemistry': [84, 56, 73, 69],

'algebra': [78, 88, 82, 87]}

#create dataframe

df\_marks = pd.DataFrame(data)

print('Original DataFrame\n------------------')

print(df\_marks)

new\_row = {'name':'Geo', 'physics':87, 'chemistry':92, 'algebra':97}

#append row to the dataframe

df\_marks = df\_marks.append(new\_row, ignore\_index=True)

print('\n\nNew row added to DataFrame\n--------------------------')

print(df\_marks)

Original DataFrame

------------------

name physics chemistry algebra

0 Somu 68 84 78

1 Kiku 74 56 88

2 Amol 77 73 82

3 Lini 78 69 87

New row added to DataFrame

--------------------------

name physics chemistry algebra

0 Somu 68 84 78

1 Kiku 74 56 88

2 Amol 77 73 82

3 Lini 78 69 87

4 Geo 87 92 97

Rename the column name:

df.rename(columns={'Parch':"ZERO",'Cabin':"Cabin\_ID"})

df\_csv = pd.read\_csv('csv\_example', names=['a', 'b', 'c'])#Assigning the name to the column

df\_csv.set\_index('age')#set the column as index

df\_csv = pd.read\_csv('csv\_example', sep=":", index\_col=1)

We can do this at the time of loading CSV file by passing a parameter called index\_col , which will automatically assign the column depicted by index\_col as a row index.

We can even provide more than one index\_col to be treated as index

df\_csv = pd.read\_csv('csv\_example', sep=":", index\_col=[0,2])

# Load Only 3 Rows  
df\_csv = pd.read\_csv('csv\_example', sep=":", nrows=3)

df\_csv = pd.read\_csv('csv\_example', skip\_blank\_lines=False, sep=":")

df.columns#Dispaly all the columns in a list format

df.rename(columns={'Parch':"ZERO",'Cabin':"Cabin\_ID"},inplace=True) #Inplace will store the modified the changes and stored it in dataframe

### Selecting columns:

* using a dot notation, e.g.

data.column\_name

,

* using square braces and the name of the column as a string, e.g.

data['column\_name']

* or using numeric indexing and the iloc selector

data.iloc[:, <column\_number>]

* square-brace selection with a list of column names, e.g.

data[['column\_name\_1', 'column\_name\_2']]

* using numeric indexing with the iloc selector and a list of column numbers, e.g.

data.iloc[:, [0,1,20,22]]

#data = data.set\_index("Area")

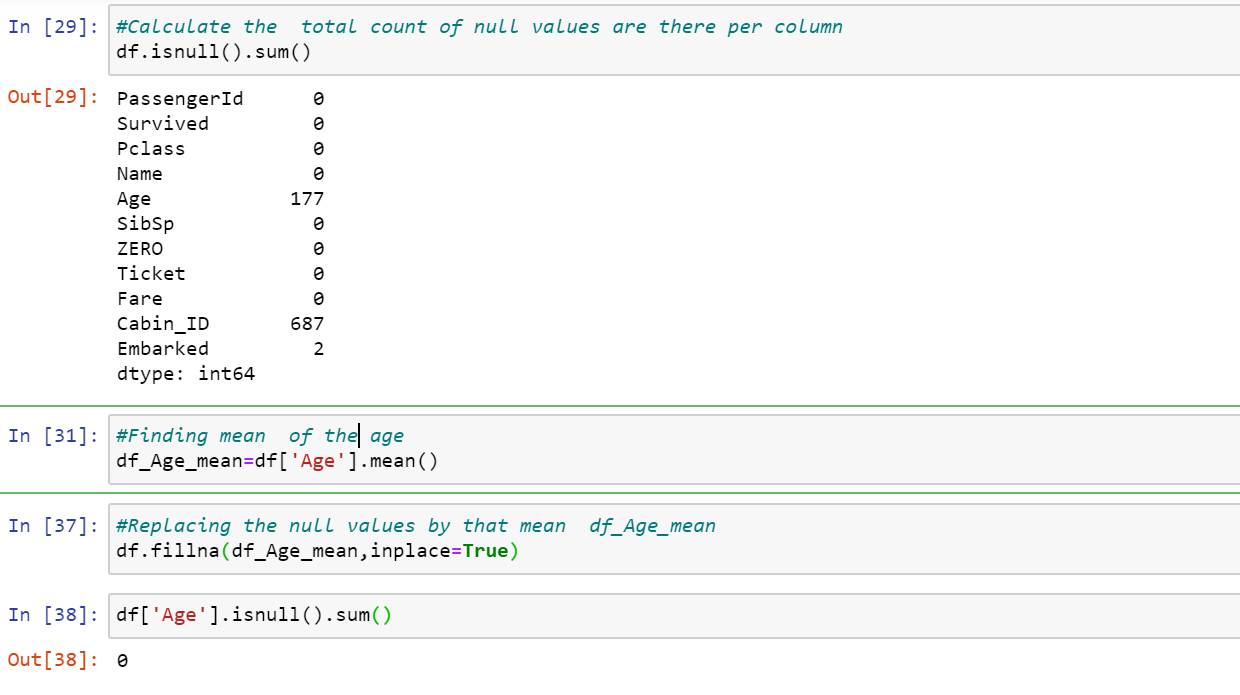
#data = data.drop("Ireland", axis=0). # Delete all rows with label "Ireland"

### How to work with missing values

There are two options in dealing with nulls:

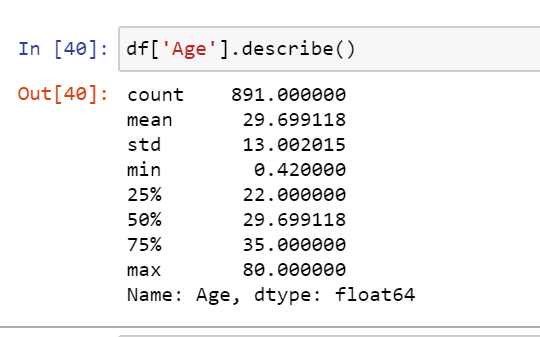
1. Get rid of rows or columns with nulls
2. Replace nulls with non-null values, a technique known as **imputation**

Let's calculate to total number of nulls in each column of our dataset. The first step is to check which cells in our DataFrame are nul



Using describe() on an entire DataFrame we can get a summary of the distribution of continuous variables:

.describe() can also be used on a categorical variable to get the count of rows, unique count of categories, top category, and freq of top category:

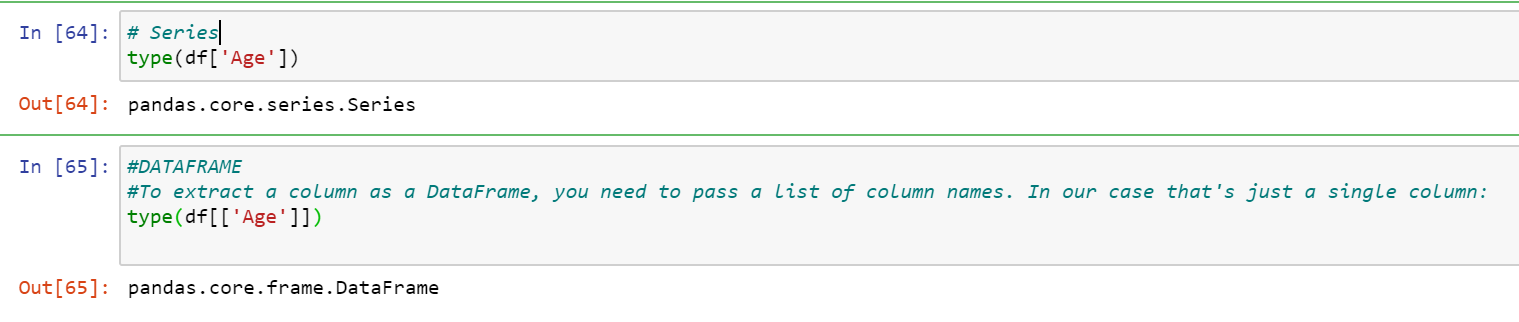


#### 

#### Relationships between continuous variables

By using the correlation method .corr() we can generate the relationship between each continuous variable:

Series vs DataFrame



### DataFrame slicing, selecting, extracting



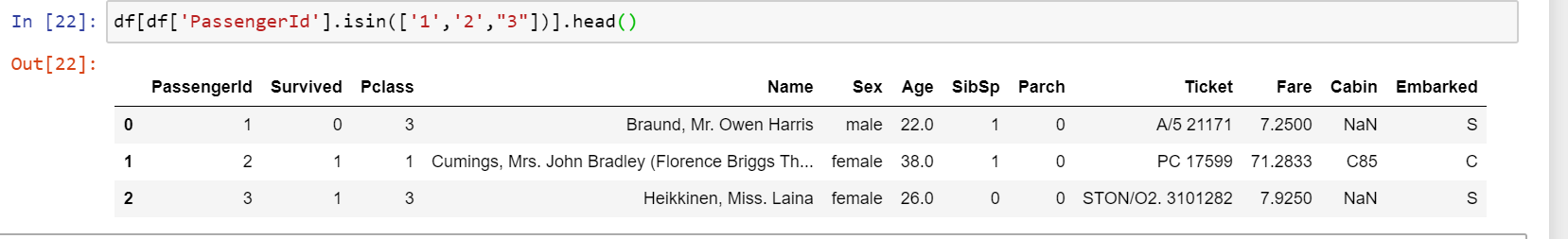
* loc gets rows (or columns) with particular *labels* from the index.
* iloc gets rows (or columns) at particular *positions* in the index (so it only takes integers).
* ix usually tries to behave like loc but falls back to behaving like iloc if a label is not present in the index.

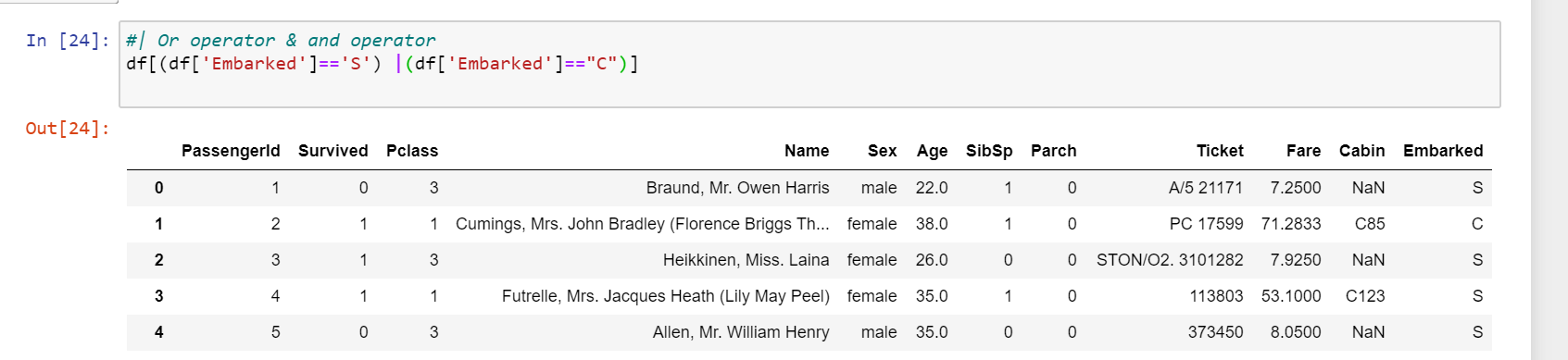
It's important to note some subtleties that can make ix slightly tricky to use:

* if the index is of integer type, ix will only use label-based indexing and not fall back to position-based indexing. If the label is not in the index, an error is raised.
* if the index does not contain *only* integers, then given an integer, ix will immediately use position-based indexing rather than label-based indexing. If however ix is given another type (e.g. a string), it can use label-based indexing.

Using the isin() method we could make this more concise though

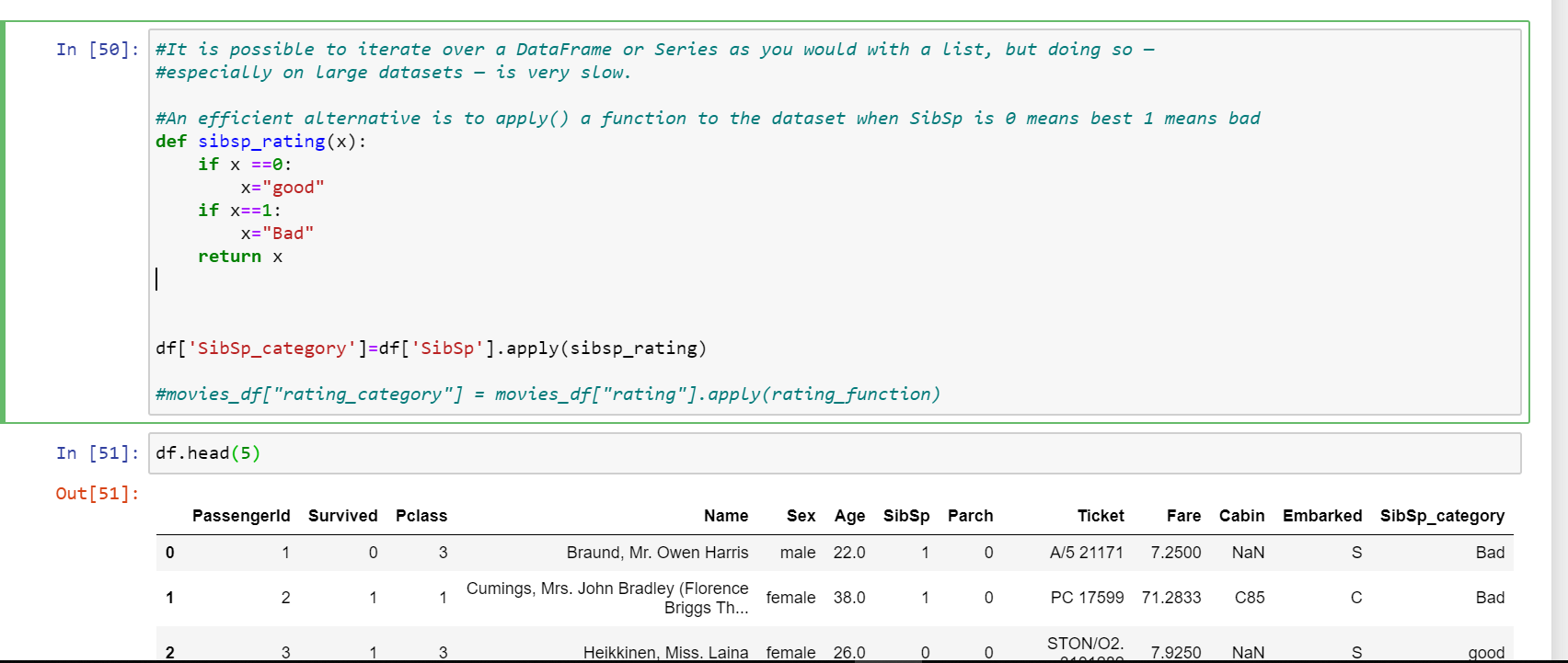
Pandas **isin()** method is used to filter data frames. isin() method helps in selecting rows with having a particular(or Multiple) value in a particular column.





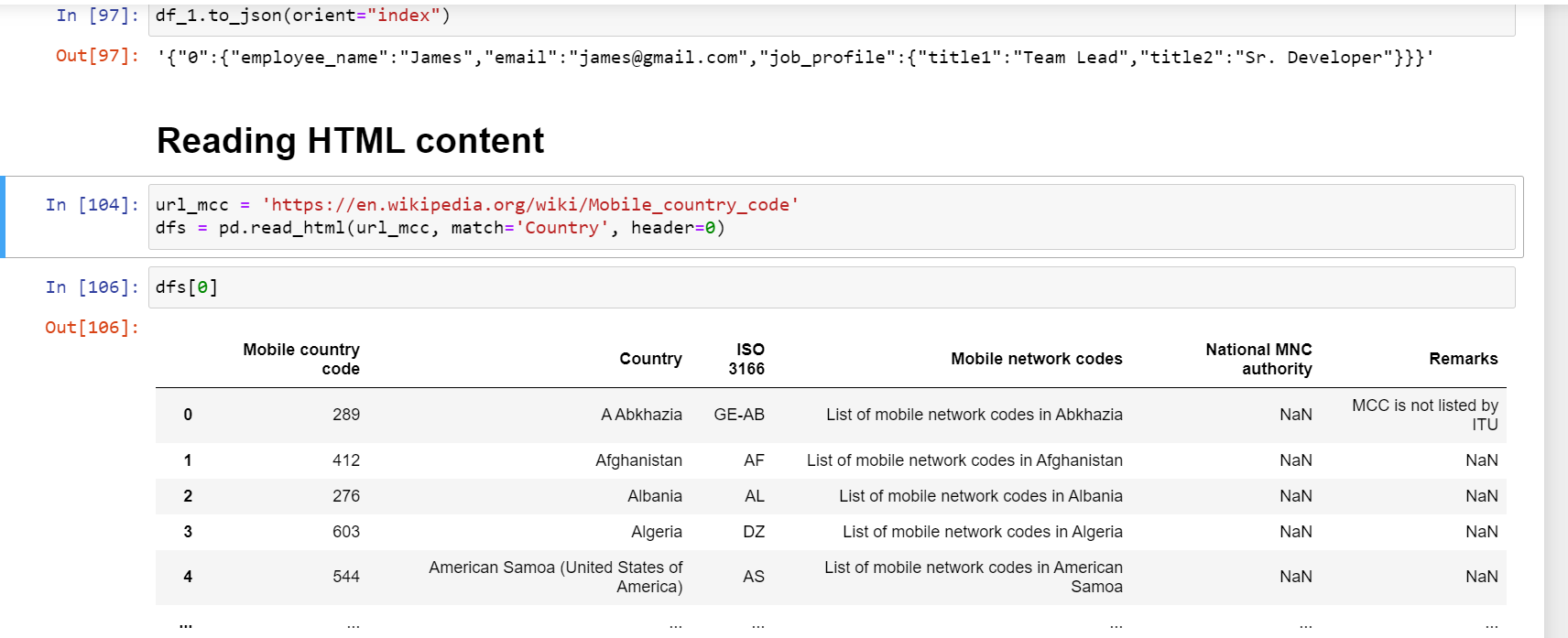


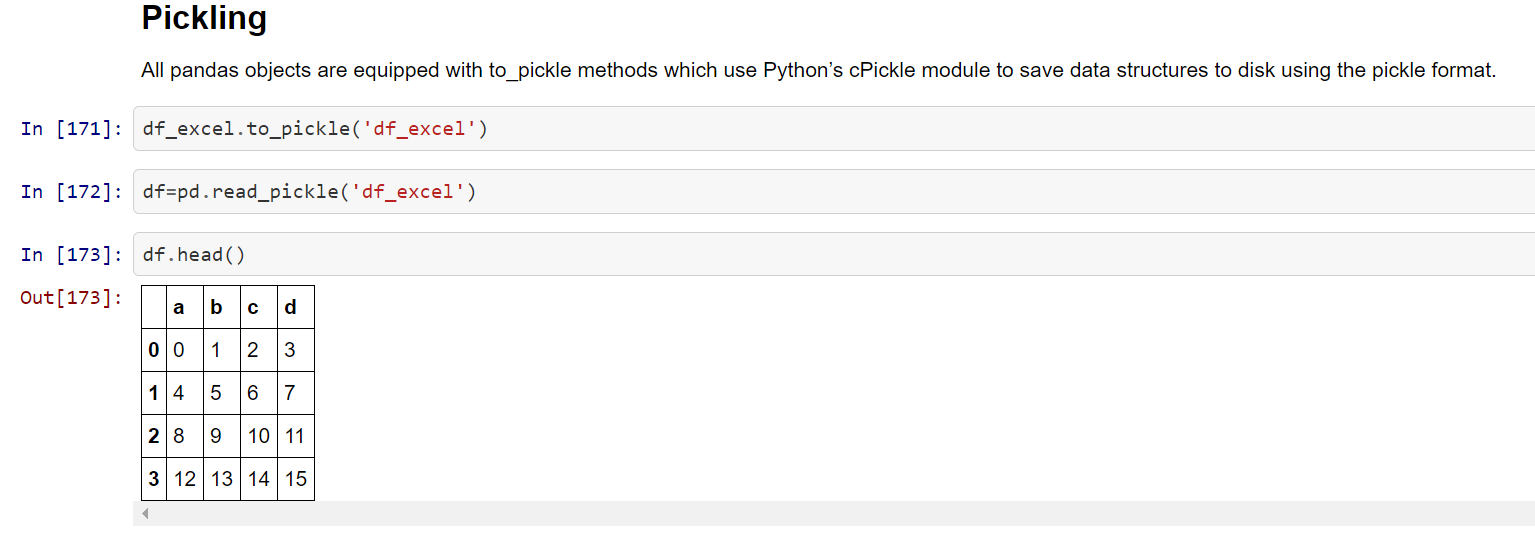
### Applying functions



using apply() will be much faster than iterating manually over rows because pandas is utilizing vectorization.







*Vectorization: a style of computer programming where operations are applied to whole arrays instead of individual elements —*[*Wikipedia*](https://en.wikipedia.org/wiki/Vectorization)

### Brief Plotting:

Another great thing about pandas is that it integrates with Matplotlib, so you get the ability to plot directly off DataFrames and Series. To get started we need to import Matplotlib (pip install matplotlib):

#### PLOTTING TIP

For categorical variables utilize Bar Charts\* and Boxplots.

For continuous variables utilize Histograms, Scatterplots, Line graphs, and Boxplots.