Statistical Analysis in Python using Pandas

Tanvi Penumudy

*In the next few minutes, we shall get '**Pandas**' covered — An extremely popular Python library that comes with **high-level data structures** and a wide range of tools for **data analysis** that every Machine Learning practitioner must be familiar with!*

Image Source: Pinterest

"Pandas aims to be the fundamental high-level building block for doing practical, real-world data analysis in Python" — [Pandas' Mission Statement]

Salient Features of the Library —

- *Fast and efficient **data manipulation** with **integrated indexing***
- ***Integrated tools** for reading/writing in various formats — CSV, text files, MS Excel, SQL, HDF5 etc.*
- *Smart **data-alignment**, integrated **handling of missing values***
- ***Flexible** in terms of **reshaping/pivoting** datasets*
- *Supports **slicing, fancy indexing** and **subsetting** of huge datasets*
- ***Size mutability***
- *High performance in **merging/joining data***
- ***Hierarchial axis indexing***
- ***Time series** functionality*
- ***Optimized performance***
- *Last but not the least, it's an **Open-Source** Python library*

> *For Frequently Asked Questions on **Pandas**, refer [Pandas Documentation]*

Getting Started with Pandas

Pandas Installation

```
pip install pandas
conda install pandas #for Anaconda
```

*Refer **pandas· PyPI** for troubleshooting*

Importing Pandas

```
import pandas as pd
```

Loading Data

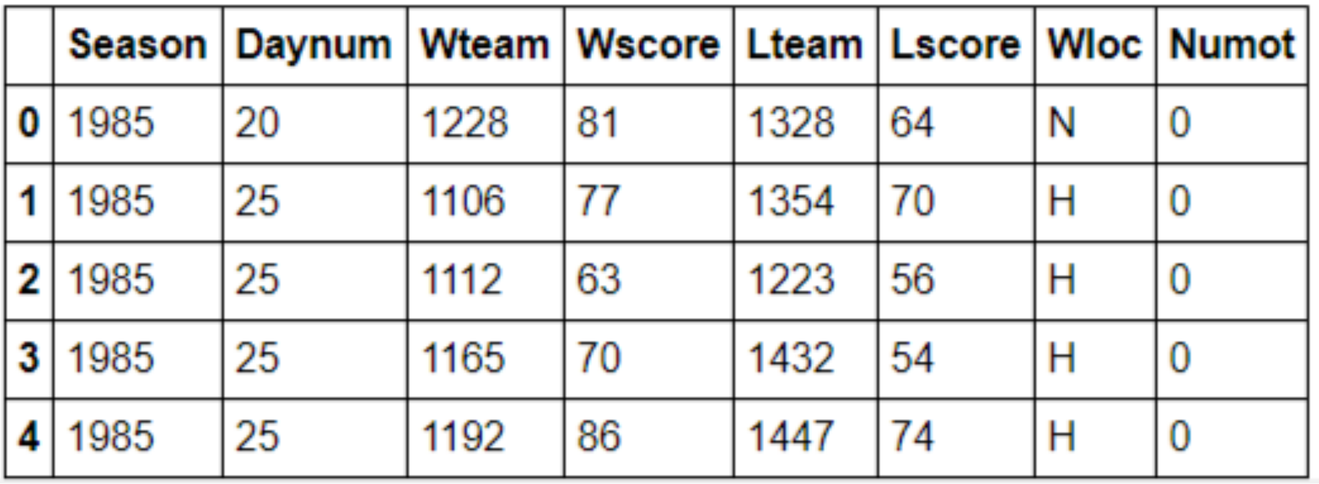
```
df = pd.read_csv('Data.csv') #Any local folder/l
```

*Find the dataset **here** — Source: **Kaggle** (Predict the 2016 NCAA Basketball Tournament)*

Useful Operations

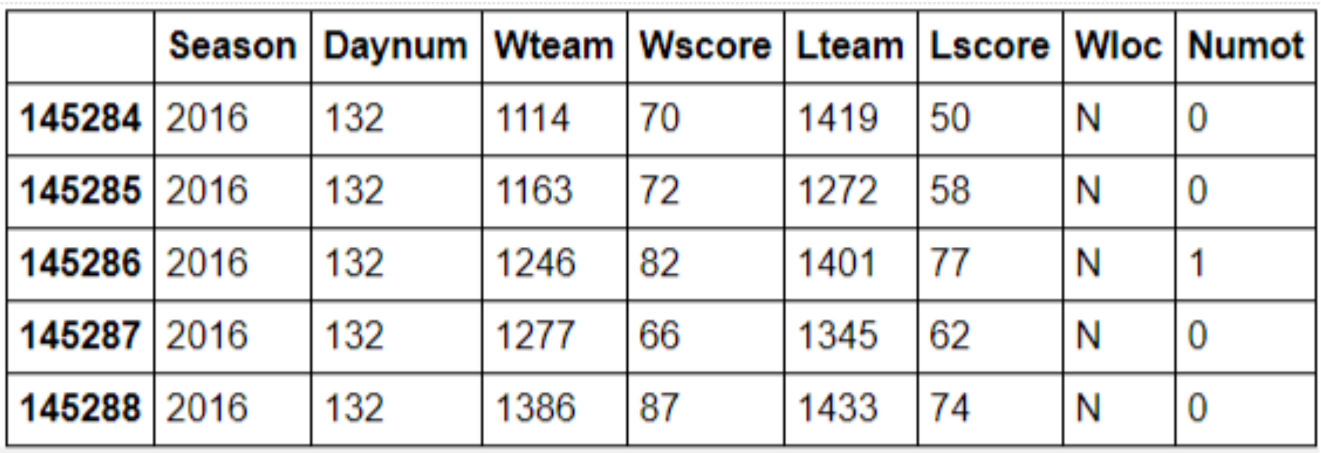
- ***head()/tail()***

```
df.head() #returns the first 5 rows of the datase
```



	Season	Daynum	Wteam	Wscore	Lteam	Lscore	Wloc	Numot
0	1985	20	1228	81	1328	64	N	0
1	1985	25	1106	77	1354	70	H	0
2	1985	25	1112	63	1223	56	H	0
3	1985	25	1165	70	1432	54	H	0
4	1985	25	1192	86	1447	74	H	0

```
df.tails() #returns the last 5 rows of the datase
```



	Season	Daynum	Wteam	Wscore	Lteam	Lscore	Wloc	Numot
145284	2016	132	1114	70	1419	50	N	0
145285	2016	132	1163	72	1272	58	N	0
145286	2016	132	1246	82	1401	77	N	1
145287	2016	132	1277	66	1345	62	N	0
145288	2016	132	1386	87	1433	74	N	0

- ***shape()***

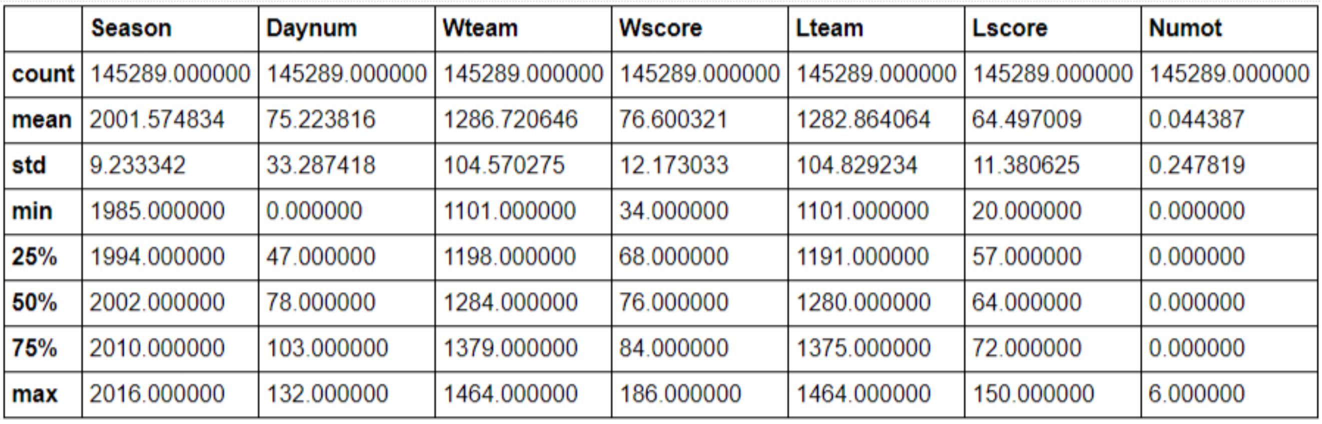
```
df.shape #returns the dimensions of the dataframe
(145289, 8)
```

- ***tolist()***

```
df.columns.tolist() #extract all the column names
['Season', 'Daynum', 'Wteam', 'Wscore', 'Lteam',
```

- ***describe()***

```
df.describe() #shows count, mean, std etc. for ea
```



	Season	Daynum	Wteam	Wscore	Lteam	Lscore	Numot
count	145289.000000	145289.000000	145289.000000	145289.000000	145289.000000	145289.000000	145289.000000
mean	2001.574834	75.223816	1286.720646	76.600321	1282.864064	64.497009	0.044387
std	9.233342	33.287418	104.570275	12.173033	104.829234	11.380625	0.247819
min	1985.000000	0.000000	1101.000000	34.000000	1101.000000	20.000000	0.000000
25%	1994.000000	47.000000	1198.000000	68.000000	1191.000000	57.000000	0.000000
50%	2002.000000	78.000000	1284.000000	76.000000	1280.000000	64.000000	0.000000
75%	2010.000000	103.000000	1379.000000	84.000000	1375.000000	72.000000	0.000000
max	2016.000000	132.000000	1464.000000	186.000000	1464.000000	150.000000	6.000000

- ***max()***

```
df.max() #returns max value for all columnsOut:
Season    2016
Daynum     132
Wteam     1464
Wscore     186
Lteam     1464
Lscore     150
Wloc         N
Numot        6
dtype: objectdf['Wscore'].max() #returns max valu
186
```

- ***mean()***

```
df['Lscore'].mean() #returns the mean of that col
64.49700940883343
```

- ***argmax()***

```
df['Wscore'].argmax() #to identify the row index(
24970
```

- ***value_counts()***

```
df['Season'].value_counts() #shows how many times
2016    5369
2014    5362
2015    5354
2013    5320
2010    5263
2012    5253
2009    5249
2011    5246
2008    5163
2007    5043
2006    4757
2005    4675
2003    4616
2004    4571
2002    4555
2000    4519
2001    4467
1999    4222
1998    4167
1997    4155
1992    4127
1991    4123
1996    4122
1995    4077
1994    4060
1990    4045
```

```
1989    4037
1993    3982
1988    3955
1987    3915
1986    3783
1985    3737
Name: Season, dtype: int64
```

Accessing Values

As per ***Pandas Documentation***, ***iloc*** is an *"integer-location based indexing for selection by position"*

```
df.iloc[[df['Wscore'].argmax()]]
#to get attributes about the game, we need to use
```

> *Let's take this a step further. Let's say you want to know the game with the highest scoring winning team (this is what we just calculated), but you then want to know how many points the losing team scored.*

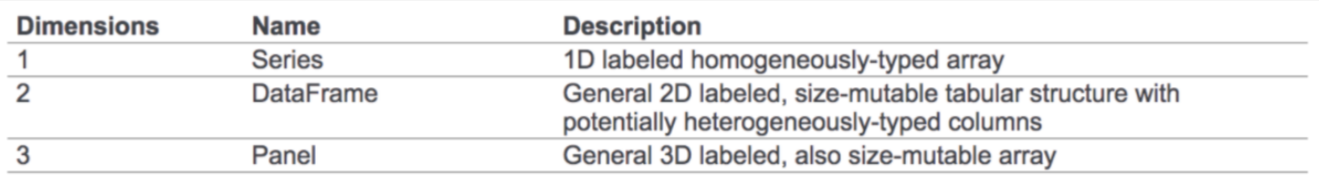
```
df.iloc[[df['Wscore'].argmax()]]['Lscore']Out:
24970    140
Name: Lscore, dtype: int64
```

When you see data displayed in the above format, you're dealing with a ***Pandas Series*** object, not a dataframe object.

```
type(df.iloc[[df['Wscore'].argmax()]]['Lscore'])
pandas.core.series.Seriestype(df.iloc[[df['Wscor
pandas.core.frame.DataFrame
```

The following is a summary of the 3 data structures in Pandas:

> *Haven’t ever really used Panels yet!*

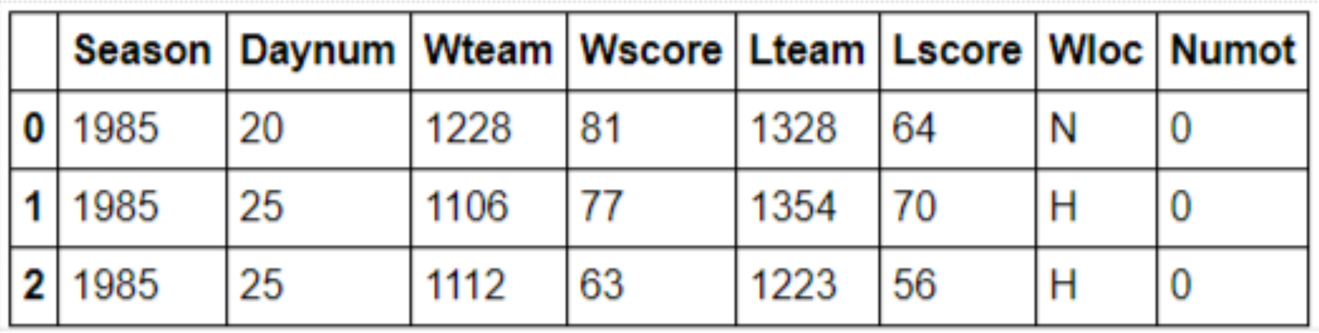


Dimensions	Name	Description
1	Series	1D labeled homogeneously-typed array
2	DataFrame	General 2D labeled, size-mutable tabular structure with potentially heterogeneously-typed columns
3	Panel	General 3D labeled, also size-mutable array

Data Structures used in Pandas

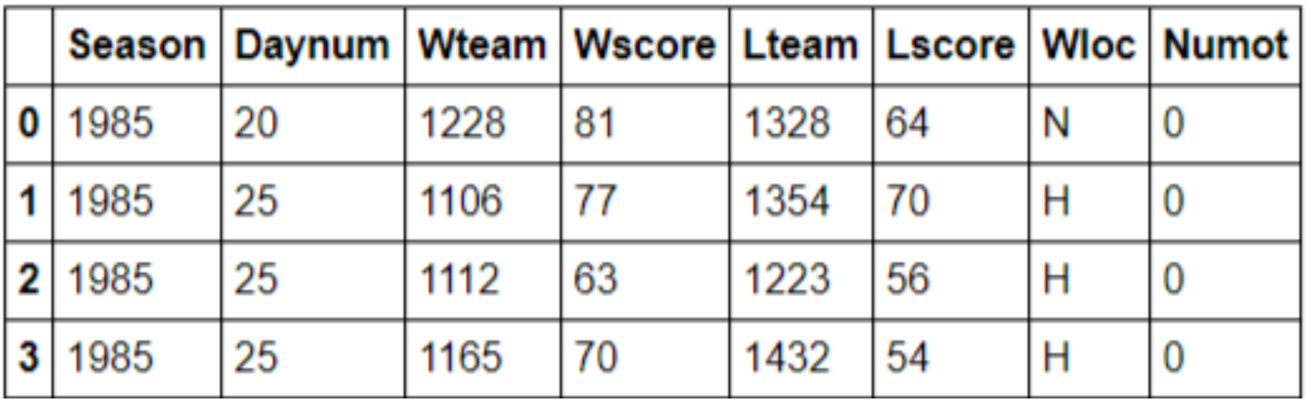
> *When you want to access values in a Series, you’ll want to just treat the Series like a Python dictionary, so you’d access the value according to its key (which is normally an integer index)*

```
df.iloc[[df['Wscore'].argmax()]]['Lscore'][24970
140df.iloc[:3]Out:
```



	Season	Daynum	Wteam	Wscore	Lteam	Lscore	Wloc	Numot
0	1985	20	1228	81	1328	64	N	0
1	1985	25	1106	77	1354	70	H	0
2	1985	25	1112	63	1223	56	H	0

```
df.loc[:3]
```



	Season	Daynum	Wteam	Wscore	Lteam	Lscore	Wloc	Numot
0	1985	20	1228	81	1328	64	N	0
1	1985	25	1106	77	1354	70	H	0
2	1985	25	1112	63	1223	56	H	0
3	1985	25	1165	70	1432	54	H	0

Notice the slight difference in that ***iloc*** is exclusive of the second number, while ***loc*** is inclusive.

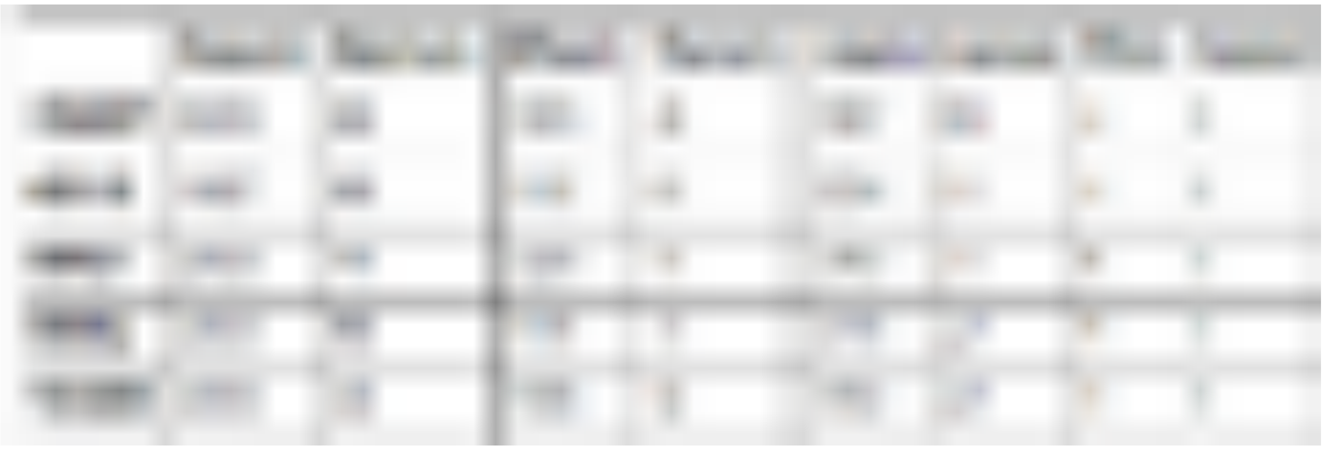
Below is an example of how you can use ***loc*** to achieve the same task as we did previously with ***iloc***.

```
df.loc[df['Wscore'].argmax(), 'Lscore']Out:
140df.at[df['Wscore'].argmax(), 'Lscore']Out:
140
```

Sorting

> *Let's say that we want to sort the dataframe in increasing order for the scores of the losing team.*

```
df.sort_values('Lscore').head()Out:
```



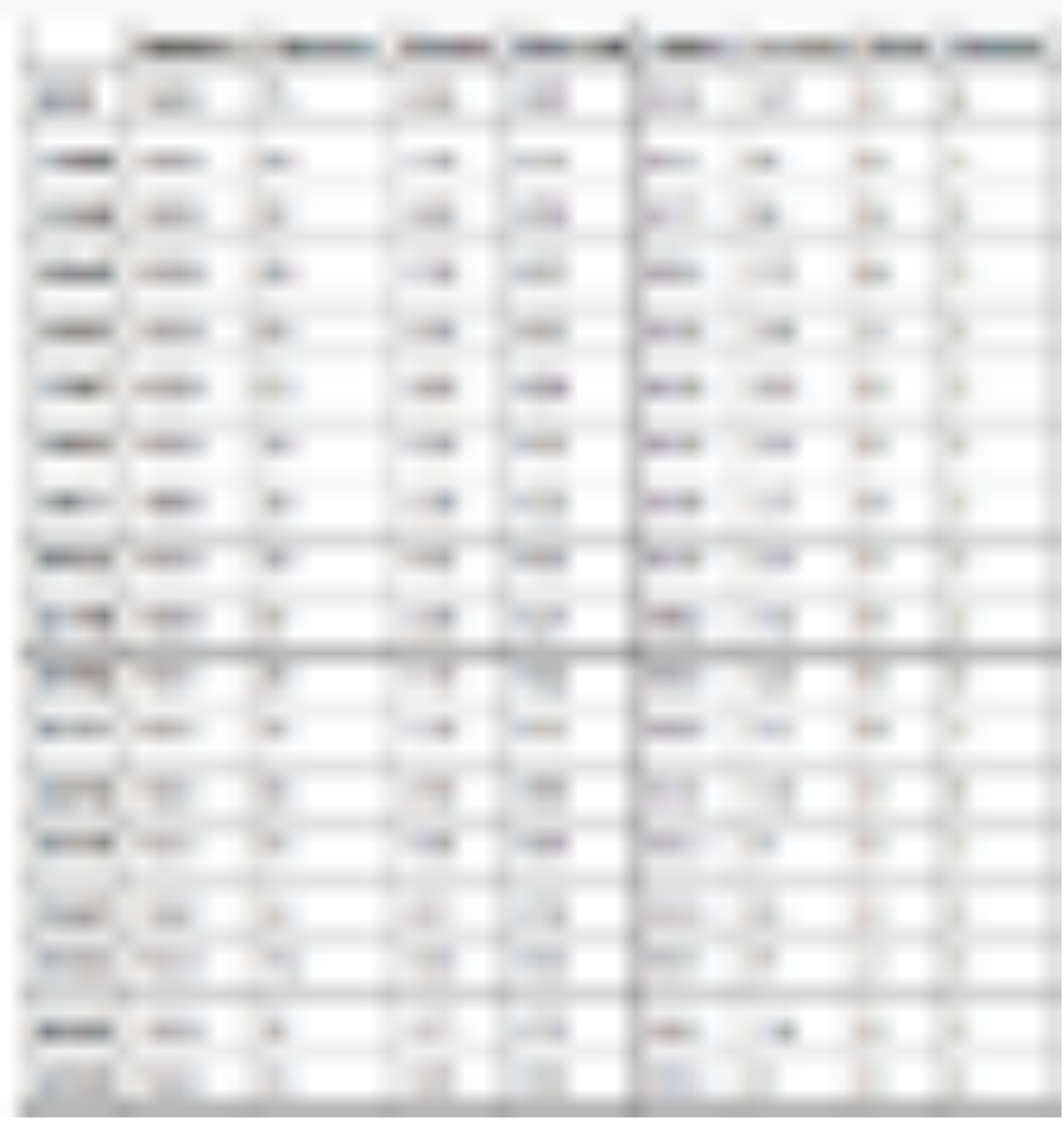
```
df.groupby('Lscore')Out:
<pandas.core.groupby.DataFrameGroupBy object at
```

Filtering Rows Conditionally

Now, let’s say we want to find all of the rows that satisfy a particular condition.

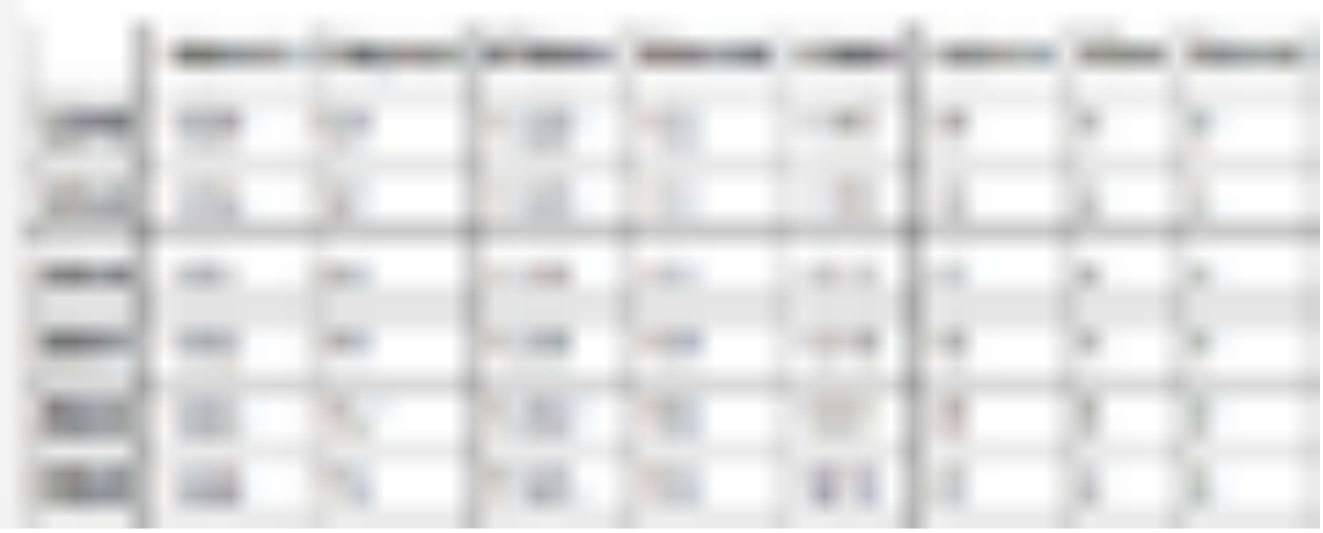
> *For example, I want to find all of the games where the winning team scored **more than 150 points.***
>
> *The idea behind this command is you want to access the column ‘Wscore’ of the dataframe **df (df[‘Wscore’])**, find which entries are above 150 **(df[‘Wscore’] > 150)**, and then returns only those specific rows in a dataframe format **(df[df[‘Wscore’] > 150])**.*

```
df[df['Wscore'] > 150]Out:
```



> *This also works if you have multiple conditions. Let’s say we want to find out when the winning team scores* ***more than 150 points*** *and when the losing team* ***scores below 100.***

```
df[(df['Wscore'] > 150) & (df['Lscore'] < 100)]Ou
```



Grouping

Another important function in Pandas is ***groupby()***. This is a function that allows you to *group entries by certain attributes* (e.g Grouping entries by Wteam number) and then *perform operations on them*.

The next command groups all the games with the *same Wteam number* and finds where *how many times that specific team won* at home, on the road, or at a neutral site.

```
df.groupby('Wteam')['Wscore'].mean().head()Out:
Wteam
1101    78.111111
1102    69.893204
1103    75.839768
1104    75.825944
1105    74.960894
Name: Wscore, dtype: float64df.groupby('Wteam')[
Wteam  Wloc
1101   H        12
       A         3
```

```
      N         3
1102  H       204
      A        73
      N        32
1103  H       324
      A       153
      N        41
Name: Wloc, dtype: int64df.valuesOut:
array([[1985, 20, 1228, ..., 64, 'N', 0],
       [1985, 25, 1106, ..., 70, 'H', 0],
       [1985, 25, 1112, ..., 56, 'H', 0],
       ...,
       [2016, 132, 1246, ..., 77, 'N', 1],
       [2016, 132, 1277, ..., 62, 'N', 0],
       [2016, 132, 1386, ..., 74, 'N', 0]], dtype
"""Now, you can simply just access elements like
1985
```

Dataframe Iteration

In order to ***iterate*** *through dataframes*, we can use the ***iterrows()*** function. Below is an example of what the first two rows look like.

> *Each row in **iterrows** is a **Series object.***

```
for index, row in df.iterrows():
    print row
    if index == 1:
        breakOut:
Season    1985
Daynum      20
```

```
Wteam     1228
Wscore      81
Lteam     1328
Lscore      64
Wloc         N
Numot        0
Name: 0, dtype: object
Season    1985
Daynum      25
Wteam     1106
Wscore      77
Lteam     1354
Lscore      70
Wloc         H
Numot        0
Name: 1, dtype: object
```

Extracting Rows and Columns

The bracket indexing operator is *one way* to extract certain columns from a dataframe.

```
df[['Wscore', 'Lscore']].head()
"""The bracket indexing operator is one way to ex
```

```
df.loc[:, ['Wscore', 'Lscore']].head()
#you can acheive the same result by using the lo
```

```
type(df['Wscore']) #difference between both opera
pandas.core.series.Seriestype(df[['Wscore']])Out
pandas.core.frame.DataFrame#only difference is th
```

> *You’ve seen before that you can access columns through **df[‘col name’]**. You can access rows by using **slicing operations**.*

```
df.iloc[0:3,:] #Here's an equivalent using ilocOu
```

Data Cleaning

The following ***isnull*** function will figure out if there are any *missing values in the dataframe*, and will then *sum up the total for each column*.

> *In this case, we have a pretty clean dataset.*

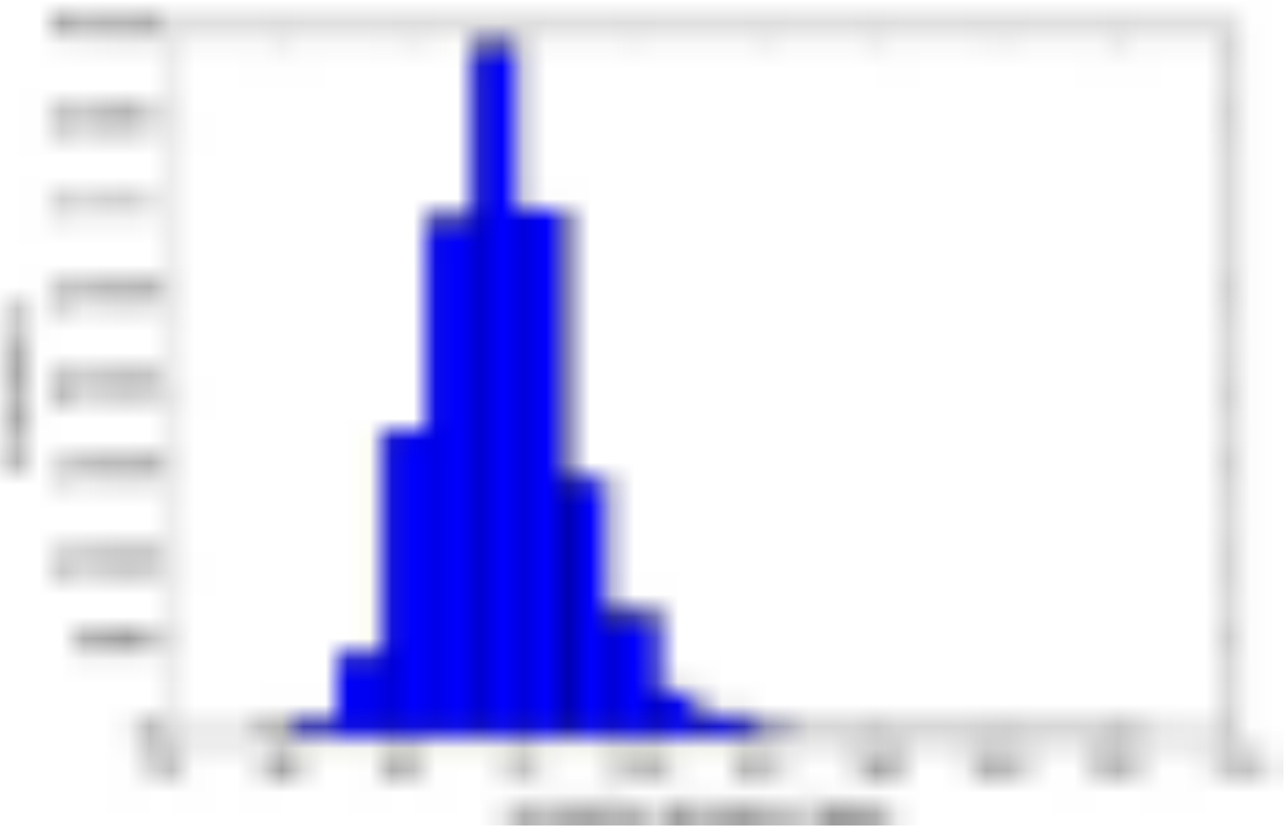
```
df.isnull().sum()Out:
Season    0
Daynum    0
Wteam     0
Wscore    0
Lteam     0
Lscore    0
Wloc      0
Numot     0
dtype: int64
```

- ***dropna()*** — This function allows you to *drop all(or some)* of the rows that have *missing values*.
- ***fillna()*** — This function allows you to *replace* the *rows* that have *missing values* with the value that you pass in.

Visualizing Data

> *An interesting way of **displaying Dataframes** is through **matplotlib**.*

```
import matplotlib.pyplot as plt
%matplotlib inline
#import matplotlib, a popular library for Data V
ax.set_xlabel('Points for Winning Team')Out:
<matplotlib.text.Text at 0x113ca8ed0>
```



Creating Kaggle Submission CSVs

> *This isn't directly Pandas related, but I assume that most people who use Pandas probably do a lot of Kaggle competitions as well.*

As you probably know, Kaggle competitions require you to create a CSV of your predictions. Here's some starter code that can help you create that csv file.

```
import numpy as np
import csv
results = [[0,10],[1,15],[2,20]]
results = pd.np.array(results)
print resultsOut:
[[ 0 10]
 [ 1 15]
 [ 2 20]]firstRow = [['id', 'pred']]
with open("result.csv", "wb") as f:
    writer = csv.writer(f)
    writer.writerows(firstRow)
    writer.writerows(results)
```

Other Useful Functions

- ***drop()*** — This function *removes the column or row* that you pass in (You also have the specify the axis).
- ***agg()*** — The ***aggregate function*** lets you compute *summary statistics* about each group.
- ***apply()*** — Lets you apply a *specific function* to any/all elements in a Dataframe or Series.
- ***get_dummies()*** — Helpful for turning *categorical data into one-hot vectors.*
- ***drop_duplicates()*** — Lets you *remove identical rows*.

Additional Resources

> *Pandas has been around for a while and there are a lot*

> *of other good resources if you're still interested in getting the most out of this library.*

- *http://pandas.pydata.org/pandas-docs/stable/10min.html*
- *https://www.datacamp.com/community/tutorials/pandas-tutorial-dataframe-python*
- *http://www.gregreda.com/2013/10/26/intro-to-pandas-data-structures/*
- *https://www.dataquest.io/blog/pandas-python-tutorial/*
- *https://drive.google.com/file/d/0BylrJAE4KMTtTUtiVExiUGVkRkE/view*
- *https://www.youtube.com/playlist?list=PL5-da3qGB5ICCsgW1MxlZ0Hq8LL5U3u9y*

Do check out my GitHub Repositories for more implementations using Pandas —

- *Statistical Analysis using Pandas Part-1*

tanvipenumudy/Winter-Internship-Internity

Repository to keep track of work assigned on a daily basis - tanvipenumudy/Winter-Internship-Internity

- *Statistical Analysis using Pandas Part-2*

tanvipenumudy/Winter-Internship-

Internity

Repository to keep track of work assigned on a daily basis - tanvipenumudy/Winter-Internship-Internity

*Also, do not forget to go through — **Pandas Documentation***