In this week, we will use basketball data downloaded from NBA.com to demonstrate how to import data into Python, how to clean up data before conducting any data analyses, as well how to describe and summarize data.

Importing data into Python

Before we import the dataset into Jupyter notebook, we need to first import the Python libraries that we will use to analyze the data

- pandas
- numpy

In [ ]:
```
import pandas as pd
import numpy as np
```

In our data repository, we have a dataset that contains NBA team information. Let's import this dataset into the Jupyter notebook.

In [ ]:
```
NBA_Teams=pd.read_csv("../../Data/Week 2/nba_teams.csv")
```

We can take a quick look at the data we imported by displaying the dataset.

In [ ]:
```
display(NBA_Teams)
```

This dataset provides some basic information of the NBA teams.

For a dataset, each row represents an observation, i.e., a team in this dataset and each column represents a variable which contains information of a characteristics of the observation. A variable can take different values in different situations. The number of observation in a dataset represents the size of our sample and the number of variables represents the richness of information in our dataset.

We can use the "shape" function in Python to see how many variables and observations in our dataset.

In [ ]:
```
NBA_Teams.shape
```

We can see that there are 30 observations (rows) and 8 variables (columns).

Renaming Variables

We can rename a variable using the "rename" function in Python.

Inplace parameter

- True: replace the old variable with a new name;
- False: create a new variable with the new name.

The first variable is unnamed, let's rename it to be "TEAM_NUMBER"; let's also rename "ID" to "TEAM_ID."

In [ ]:
```
NBA_Teams.rename(columns={'Unnamed: 0':'TEAM_NUMBER'}, inplace=True)

NBA_Teams.rename(columns={'ID':'TEAM_ID'}, inplace=True)
display(NBA_Teams)
```

Self Test - 1

- Rename "FULL_NAME" to "TEAM_NAME"

In [ ]:
```
#Your Code Here
```

Dropping Variables (columns)

To drop a variable, i.e., to delete a column, we can use the "drop" command.

- We need to provide the name of the variable;
- We also need to use the argument "axis=1" which tells Python that we are dropping a column, not a row.

The variable "TEAM_NUMBER" has little meaning, let's drop it.

In [ ]:
```
NBA_Teams.drop(['TEAM_NUMBER'], axis=1, inplace=True)
display(NBA_Teams)
```

Next we will work on game level data.

Import the game level dataset from our data repository.

- We can display just first five rows of the dataset using the "head" command.

In [ ]:
```
Games=pd.read_csv("../../Data/Week 2/basketball_games.csv")
Games.head()
```

Upon importing the game data, we notice that the first five games are not NBA games, instead, they are WNBA games. Indeed, this dataset contains NBA games, WNBA games, NBA 2K (simulation video) games.

Dropping observations (rows)

To drop an observation, we can use the index number on the left to specify the row we want to drop.

- The argument axis=0 specifies that we want to drop a row instead of a column.

In [ ]:
```
Games.drop([0], axis=0, inplace=True)
Games.head()
```

More often, we will drop observations based on certain conditions.

For example, Las Vegas Aces is a women's basketball team. If we are only going to focus on men's basketball games, we will drop all the games played by Las Vegas Aces. In this case, we don't have to use the "drop" function. We can specify our TEAM_NAME variables to be not equal to "Las Vegas Aces."

In [ ]:
```
Games=Games[Games.TEAM_NAME !="Las Vegas Aces"]
Games.head()
```

Self Test - 2

- Drop all the Phoenix Mercury games

In [ ]:
```
#Your Code Here
```

Merging Dataframes

We will only focus on NBA games. We could merge the NBA_Teams and Games datasets to filter out NBA games.

Teams are identified by the TEAM_ID. So, let's merge the datasets by TEAM_ID. Since the variable "TEAM_NAME" is also present in both datasets, we could also include this variable as a criteria to merge the datasets so that in our new dataset, there is no duplicate variables.

In [ ]:
```
NBA_Games=pd.merge(NBA_Teams, Games, on=['TEAM_ID', 'TEAM_NAME'])
NBA_Games.head()
```

Understanding and cleaning the merged dataset

As you can tell, the merged dataset has a lot more variables and Python cannot fit all of them in the screen.

We can obtain the list of variables using the "columns" command. This provides us a full list of variables in our dataset.

In [ ]:
```
NBA_Games.columns
```

Data Cleaning

The variable "ABBREVIATION" AND "TEAM_ABBREVIATION" carry the same information and it is not necessary to keep both of them.

- Delete "ABBREVIATION"

In [ ]:
```
NBA_Games.drop(['ABBREVIATION'], axis=1, inplace=True, errors='ignore')
```

Self Test - 3

- Find the number of observations and the number of variables in the dataset

In [ ]:
```
#Your Code Here
```

The merged dataset is sorted by the criteria we use to merge the datasets. Thus, the NBA_Games dataset is currently sorted by "TEAM_ID." We may be interested to sort the data by other criteria, for example, the date of the game.

We can do so by using the "sort_values" option.

In our dataset, "GAME_ID" is created based on the date of the game. We can sort the games by "GAME_ID" and display the 20 most recent games.

In [ ]:
```
NBA_Games.sort_values(by=['GAME_ID'], ascending=[False]).head(20)
```

Missing Values

Before we move on to doing any data analyses, we usually need to check if there is any missing value, that is, the source may have failed to collect some information.

We can use the info() command which will return the total number of observations that have real values. By looking at these total numbers, we can see if there is any variable with missing value.

In [ ]:
```
NBA_Games.info()
```

*The total number of rows is 18956, so there are missing values in variable WL, FG*PCT, FG3_PCT, and FT_PCT._

Detecting missing values

We can use the isnull() function and the notnull() function to detect where the missing values are.

In [ ]:
```
NBA_Games.notnull()
```

Handling Missing Values

There are two main approaches to handle missing values.

- First, we can simply drop the observations with missing value.

Drop observations with missing value in the variable "FG_PCT"

In [ ]:
```
NBA_Games=NBA_Games[pd.notnull(NBA_Games["FG_PCT"])]
NBA_Games.shape
```

- Second, we can replace the missing values with valid values (Imputation), such as mean and median.

We can use the fillna() command to replace missing values with the mean or the median of the variable.

In [ ]:
```
NBA_Games=NBA_Games.fillna(NBA_Games.mean())
NBA_Games.info()
```

Creating variables

We can create a variable equals to the total number of goals made.

In [ ]:
```
NBA_Games['GM']=NBA_Games['FGM']+NBA_Games['FG3M']+NBA_Games['FTM']
```

Self Test - 4

- Create a variable called "GA" equals to the total number of goals attempted.

In [ ]:
```
#Your Code Here
```

Create variables based on conditions

- We can create a variable conditional on the value of another variable.

For example, we can create a variable "RESULT" that equals to 1 if the team won the game and 0 otherwise. The result of the game can be captured in the points of the team receive, whether it was positive or negative.

In [ ]:
```
NBA_Games['RESULT'] = np.where(NBA_Games['PLUS_MINUS']>0, 'W', 'L')
```

We will now drop this newly created "RESULT" variable.

In [ ]:
```
NBA_Games.drop(['RESULT'], axis=1, inplace=True)
```

Create a variable within group

In the dataset, each game has two observations, one represents the statistics of the home team, one represents those of the away team. Both observations have the same GAME_ID. We can create a variable "POINT_DIFF" that equals the difference between the points earned by the two teams.

We will first sort the data not only by the "GAME_ID" but also by the result "WL".

In [ ]:
```
NBA_Games.sort_values(['GAME_ID','WL'], inplace=True)
NBA_Games["POINT_DIFF"]=NBA_Games.groupby(["GAME_ID"])["PTS"].diff()
```

The "POINT_DIFF" variable only has the point difference for the winning team, we need to impute the point difference for the losing team as well.

In [ ]:
```
NBA_Games['POINT_DIFF'] = NBA_Games['POINT_DIFF'].fillna(NBA_Games.groupby('GAME_ID')['POINT_DIFF'].transform('mean'))
```

- We can also drop all observations with missing value in at least one variable using the "dropna()" command.

In [ ]:
```
NBA_Games=NBA_Games.dropna()
NBA_Games.shape
```

Creating new dataframe

Create a new dataframe that aggregates information by group

Sometimes we may want to work with season level data rather than team level data. We can create a new dataset that includes aggregate information of team statistics in each season.

In [ ]:
```
NBA_Team_Stats=NBA_Games.groupby(['TEAM_ID', 'SEASON_ID'])['PTS','FGM','FGA','FG_PCT','FG3M','FG3A','FG3_PCT','FTM','FTA','FT
display(NBA_Team_Stats)
```

Notice that the newly created dataset has two levels of index, the "TEAM_ID" and "SEASON_ID"

If we want to convert these two indexes back as variables, we can use the "reset_index" command.

In [ ]:
```
NBA_Team_Stats=NBA_Team_Stats.reset_index()
display(NBA_Team_Stats)
```

We can create a variable that equals to the total number of observations within a specified group using the size() command.

- Create a variable that equals to the total number of games played by a team in each season, name this variable "GAME_COUNT".

In [ ]:
```
NBA_Game_Count=NBA_Games.groupby(['TEAM_ID','SEASON_ID']).size().reset_index(name='GAME_COUNT')
display(NBA_Game_Count)
```

Saving data

We can save a dataframe by exporting the edited dataframe to csv file using the “to_csv” command.

- Save merged data as a csv file *We can use the "index=False" command to save the data without adding the index as a column in the csv file*

In [ ]:
```
NBA_Games.to_csv("../../Data/Week 2/NBA_Games.csv", index=False)
```

In [ ]:
```

```