pandas: Analyzing Basketball Data
Mathematical Programming with Python

MATH 2604: Advanced Scientific Computing 4
Spring 2025
Monday/Wednesday/Friday, 1:00-1:50pm

https://people.sc.fsu.edu/~jburkardt/classes/python_2025/basketball/basketball.pdf

"Basketball Data"

- *we must read a Comma Separated Variable (csv) file of data;*
- *the file records information about 26 basketball players;*
- *read this data into a* `pandas` *DataFrame;*
- *use player names as the index;*
- *print a row, a column and an entry of the data;*
- *add one player to the roster;*
- *add a new "AveragePoints" category;*
- *correct one item of data;*
- *compute minimum, mean, and maximum of some data;*
- *search for correlations between data items;*
- *scatterplot some correlated data;*
- *make linear model of correlated data;*
- *write our corrected information to a new csv file;*

1 The basketball data

We have a file containing information about a (fictional) basketball team. There are 26 members of the team. This data includes the player's number, name, position, height, weight, earnings from sponsorships, whether they have a shoe sponsorship, their career stage, and age. Notice that this information includes numbers, strings, and boolean yes/no variables. This information is stored in a file. Each line of the file lists the data for a particular player. The very first line labels the information in each column. Because the data is not simply numeric, it is necessary to use a format that can handle mixed data types. Our data is stored using the Comma Separated Variable `csv` format. That is, the data items on a given line are separated by commas.

In some cases, a `csv` file requires that each string be a single word (no spaces), or that strings be quoted. In some `csv` files, the data is separated by spaces or tab characters.

We will use the first line of our file to identify the columns of data. We will use the second item (column 1) of our data, the player name, to identify each row.

Luckily, a `csv` file is a text file, and can be printed or viewed with an editor. For that reason, we can take a look at the first few lines of text to see what we are dealing with:

```
ID,Name,Position,Height,Weight,SponsorshipEarnings,ShoeSponsor,CareerStage,Age
1,Adam,forward,192,218,561000,yes,veteran,29
2,Bill,center,218,251,60000,no,mid-career,35
3,Clay,forward,197,221,1312000,no,rookie,22
```

This is the file which we will want to work with during this exercise.

2 Reading a csv file into a DataFrame

For our exercise today, we first need to import the `pandas` library with the conventional abbreviation:

```
import pandas as pd
```

In our `matplotlib2` discussion, we worked one time with a `csv` file, using the `numpy` function called `genfromtxt()`. Because we want our data to be read into a `pandas DataFrame`, we will use the function `pd.read_csv()` instead.

Of course, we need to specify the name of the file we are reading. Along with that, there are many (in fact more than 70) optional arguments we can specify here, because the style of `csv` files varies so much. We are mainly concerned with two issues:

- yes, this file includes an initial header line of column labels;
- we want to use the names (column 1) as row indices;

We can accomplish this, and store the information in a DataFrame called `data`, by

```
data = pd.read_csv ( 'basketball_data.csv', header = 0, index_col = 1 }
```

The argument `header = 0` actually means that the labels are already in the file, in row 0. The argument `index_col = 1` says that we should use column 1 (the player names) for the row indentifiers.

If this command is executed properly, we now have a `DataFrame` named `data` containing the basketball information.

3 Viewing the DataFrame

Because data files often contain an enormous amount of information, `pandas` offers a way to print just a selected view with the `head()` method:

```
print ( data.head() )
```

However, we can go ahead and try to print everything. By default, we will get all the rows of information, but only the first and last few columns:

```
print ( data )
```

For some reason, `pandas` really prefers not to show all the columns. Sometimes this can be annoying, and you can try to insist on seeing everything by using the `.to_string()` method:

```
print ( data.to_string() )
```

4 Examining selected entries of a DataFrame

As we requested, the data is indexed by player name. You can see the list by

```
print ( data.index )
```

The player names can be used, along with the `loc()` function to specify a particular row of information.

```
print ( data.loc['Wade'] )
```

If we wish to specify a column of data, then we have to use a double index with `loc()`, letting the first index be a colon:

```
print ( data.loc[:,'Height'] )
```

For a single specify entry, we list both the player name and the column label:

```
print ( data.loc['Pete','Weight'] )
```

If we want to print all the players more than 200 centimeters tall, we can do that this way:

```
print ( data.loc[200 < data['Height'] ] )
```

If we want to print all the rookie players:

```
print ( data.loc[ data['CareerStage']=='rookie' ] )
```

5 Adding a row to a DataFrame

A new player, **Zorg**, has joined the team, and so we need to add a new row to our `DataFrame`. As long as we have all the other information to hand, we can easily enter this as well.

```
data.loc['Zorg'] = [ 27, 'center', 190, 210, 0, 'no', 'rookie', 24 ]
```

6 Adding a column to a DataFrame

We could also add a new column to our `DataFrame`. We simply specify a new label, and a list of the value for each player. Note that because of our new player Zorg, we now have 27 values to enter:

```
data['PointAverage'] = [ \
   0.2,  5.0,  6.3,  5.5,  4.6,\
   3.3,  4.1,  0.8,  0.8,  5.3,\
   3.8,  1.8,  3.2,  4.1,  1.6,\
   7.6,  1.5,  0.3,  2.1,  2.9,\
   4.3,  2.2,  2.5,  1.0,  4.8,\
   1.5,  3.6 ]
```

We can easily determine the average point average, and the average point totals:

```
average_average = data.loc[:,'PointAverage'].mean ( )
point_total = data.loc[:,'PointAverage'].sum ( )
```

It would also be interesting to find the point averages for the three positions. One way to do this is to pull out all the records for the given position as a new DataFrame, and get the point average mean from this special set:

```
forward = data.loc[data['Position'] == 'forward']      # select forwards
point_forward = forward.loc[:,'PointAverage'].mean()   # average points
print ( '' )
print ( '  In an average game, an average forward makes = ', point_forward )
```

7 Reporting statistical data

We can request the minimum, mean, maximum, standard deviation or variance of the data in any column which is numeric. To get the minimums, our command would be

```
data.min ( numeric_only = True )
```

A very interesting question is to what extent one data column is correlated with another. This is somewhat similar to taking the dot product of two vectors. If both items tend to rise or fall together, their correlation will be close to +1; if one goes up when the other goes down, this value will be close to -1. If one variable seems to have little relation with the other, the correlation will be close to 0.

We can compute the correlation of all pairs of numeric data items. The result forms a correlation matrix which we might refer to as `C`. Clearly, `C[i,i]=1`, `C[i,j]=C[j,i]` for all items i and j. We are most interested in entries which are close to ± 1, since these are reporting that there is some relationship between the two variables.

Our request for a correlation matrix is written

```
data.corr ( numeric_only = True )
```

Again, `pandas` thinks the array is too large to show completely, and again, we can override this with the `.to_string()` method:

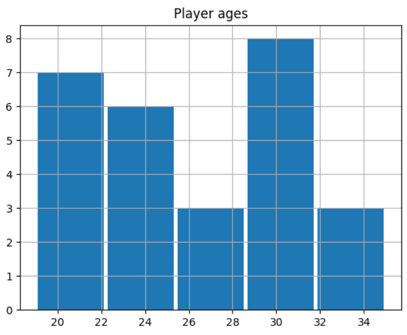
```
data.corr ( numeric_only = True ).to_string()
```

There are as least two pairs of data that show a strong relationship: height and weight (we could have guessed), but also age and sponsorship earnings. We will make plots to illustrate these relationships.

8 Histograms

We can make a histogram of the player ages.

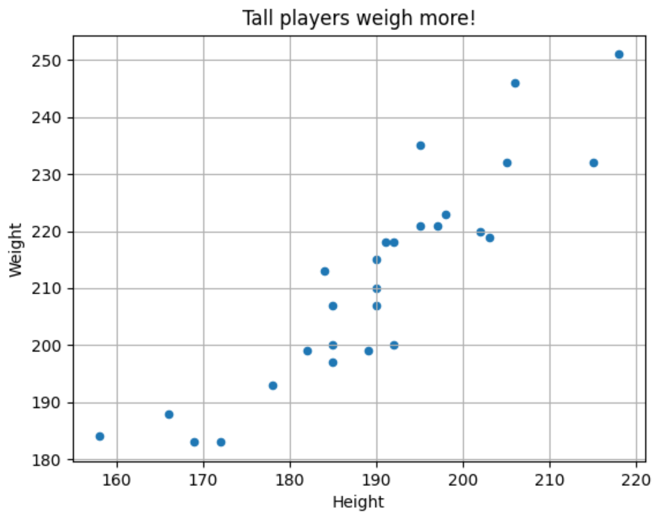
```
data.hist ( column = 'Age', bins = 5, rwidth = 0.95 )
plt.title ( 'Player ages' )
plt.show ( )
```



9 Scatterplots

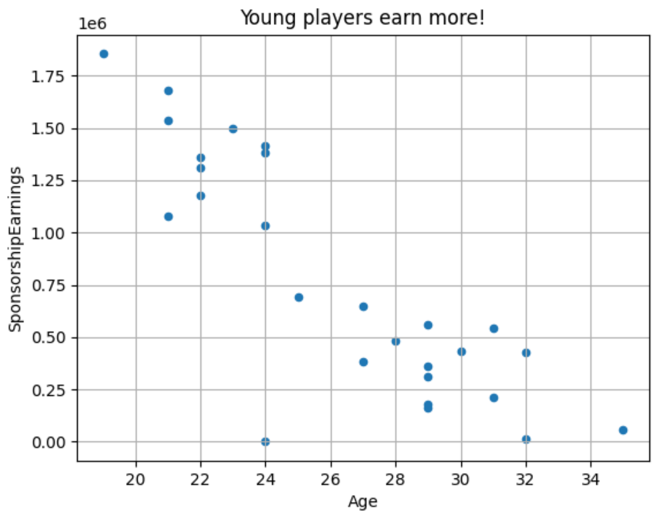
To see how strong the relationship is between height and weight, we give the commands:

```
data.plot.scatter ( 'Height', 'Weight' )
plt.show ( )
```



For age versus earnings, we have

```
data.plot.scatter ( 'Age', 'SponsorshipEarnings' )
plt.show ( )
```



In both cases, we can see that the data follows a roughly linear pattern, in one case with positive correlation (rising together), and in the other, negative correlation (earnings fall as age increases).

10 Using linear regression

The two relationships that we have discovered both seem to be roughly linear. Thus, we would be interested in being able to display a plot in which a line approximates the trend of the data, and we might even want to know the values of a and b in the approximating linear relationship $y = ax + b$.

The linear least squares solution to our problem finds values such that we minimize the sum of the squared errors,

$$E(a,b) = \sum_{i=0}^{i<n} (ax_i + b - y_i)^2$$

The solution to this problem can be found by setting to zero the partial derivatives $\frac{\partial E}{\partial a}$ and $\frac{\partial E}{\partial b}$ and doing some manipulation to find

$$a = \frac{x \cdot (y - \bar{y})}{x \cdot (x - \bar{x})}$$
$$b = \bar{y} - a * \bar{x}$$

where the barred quanties are mean values, and the dot indicates a vector dot product. This can be programmed as:

```
def llsq ( x, y ):
  import numpy as np
  xbar = np.mean ( x )
  ybar = np.mean ( y )

  xy = np.dot ( x, y - ybar )
  xx = np.dot ( x, x - xbar )

  a = xy / xx
  b = ybar - a * xbar

  return a, b
```

11 Displaying a linear approximation to data

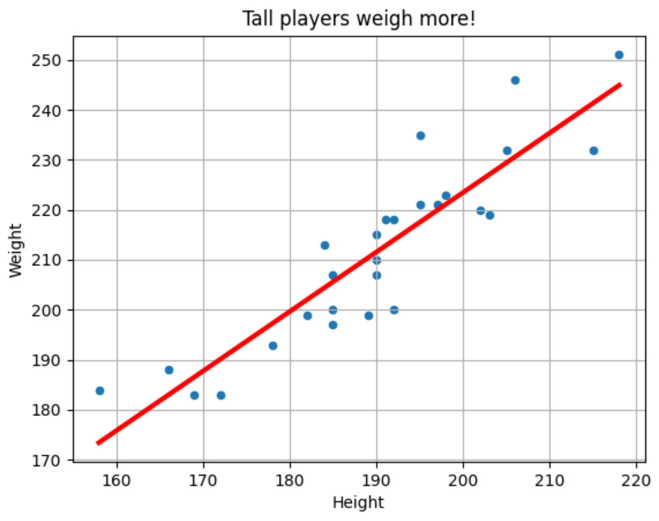
We can see how strong our linear relationship is by plotting it against our data.
For the height and weight relationship, we have:

```
  import matplotlib.pyplot as plt

  x = data['Height']
  y = data['Weight']

  a, b = llsq ( x, y )
  xmin = np.min ( x )
  xmax = np.max ( x )
  ymin = a * xmin + b
  ymax = a + xmax + b

  data.plot.scatter ( 'Height', 'Weight' )
  plt.plot ( [ xmin, xmax ], [ ymin, ymax ] 'r-', linewidth = 3 )
  plt.show ( )
```



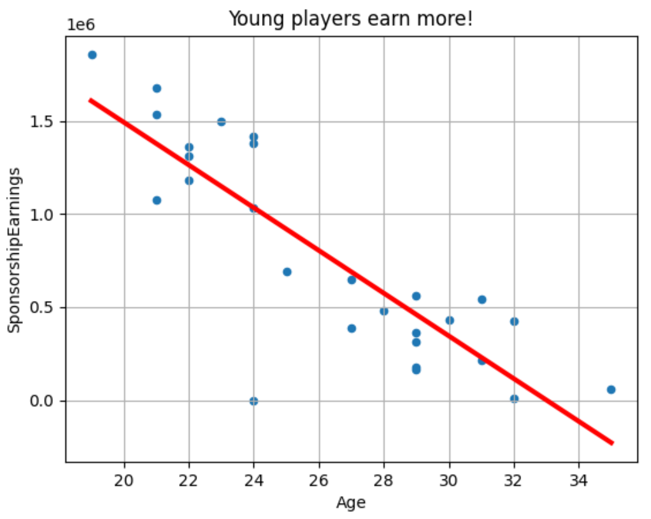
For the age and sponsorship, we have:

```
import matplotlib.pyplot as plt

x = data['Age']
y = data['SponsorshipEarnings']

a, b = llsq ( x, y )
xmin = np.min ( x )
xmax = np.max ( x )
ymin = a * xmin + b
ymax = a + xmax + b

data.plot.scatter ( 'Age', 'SponsorshipEarnings' )
plt.plot ( [ xmin, xmax ], [ ymin, ymax ] 'r-', linewidth = 3 )
plt.show ( )
```



12 Writing a DataFrame to a csv file

Because of the corrections we made to our data, we may wish to save a new `cvs` file, perhaps with the name *basketball2_data.csv*. We can do that easily with the `to_csv()` function. While there are options available that allow us to store just some of the rows and columns, the default call is probably all we need:

```
data.to_csv ( 'basketball2_data.csv' )
```

And this ends our exploration of the basketball data file!