Contents

Introduction

Lab materials adapted from the UC Berkely Data 6

The objectives for this lab are to, learn more about:

- Review of Histograms
- Scatter Plots
- Customizing Scatter Plots
- Line Plots
- Multiple Line Plots

working with lab12.blank.ipynb

Data

Today's first data set comes from Basketball Reference. It contains per-game averages of players in the 2019-2020 NBA season.

A description of each column:

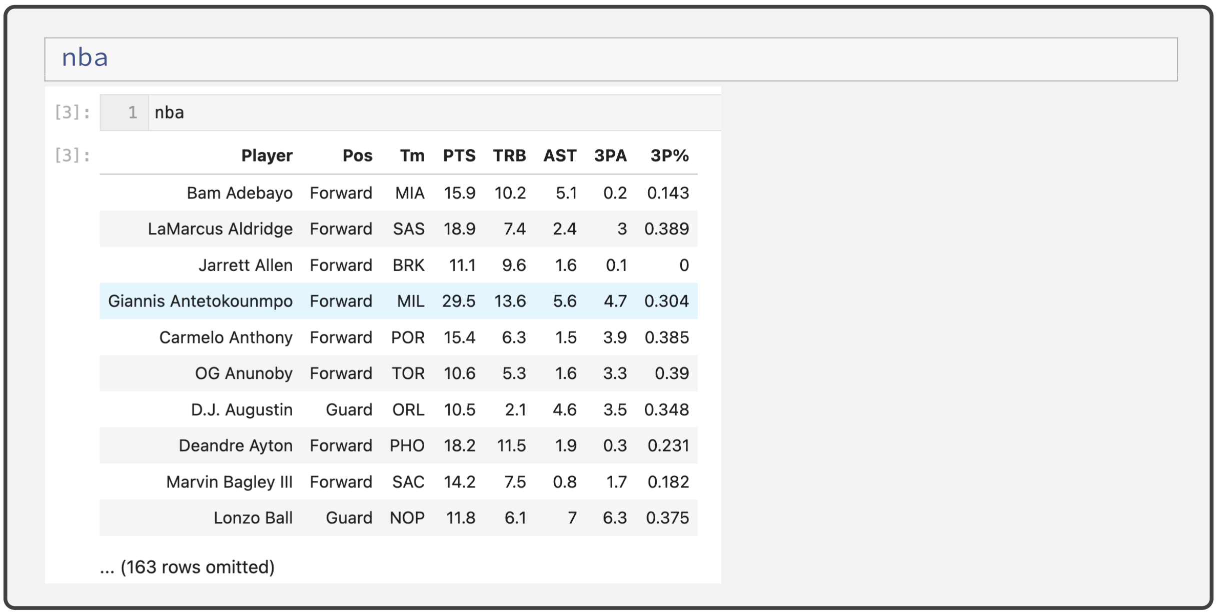
- `'Player'`: name
- `'Pos'`: general position (either Forward or Guard)
- `'Tm'`: abbreviated team
- `'PTS'`: average number of points scored per game
- `'TRB'`: average number of rebounds per game (a player receives a rebound when they grab the ball after someone misses)
- `'AST'`: average number of assists per game (a player receives an assist when they pass the ball to someone who then scores)
- `'3PA'`: average number of three-point shots attempted per game (a three point shot is one from behind a certain line, which is between 22-24 feet from the basket)
- `'3P%'`: average proportion of three-point shots that go in

You can see in the code that loads the data into a Table, `nba`, that the columns are limited to those listed above.

Also, the rows (players) are filtered where they have an average number of three-point shots per game not equal to 0.

Additionally, the rows are filtered where the player's position must be a guard or forward.

Finally, the rows are filtered where only players who averaged at least 10 points per game in the season are included.



nba

```
[3]: nba
```

[3]:

Player	Pos	Tm	PTS	TRB	AST	3PA	3P%
Bam Adebayo	Forward	MIA	15.9	10.2	5.1	0.2	0.143
LaMarcus Aldridge	Forward	SAS	18.9	7.4	2.4	3	0.389
Jarrett Allen	Forward	BRK	11.1	9.6	1.6	0.1	0
Giannis Antetokounmpo	Forward	MIL	29.5	13.6	5.6	4.7	0.304
Carmelo Anthony	Forward	POR	15.4	6.3	1.5	3.9	0.385
OG Anunoby	Forward	TOR	10.6	5.3	1.6	3.3	0.39
D.J. Augustin	Guard	ORL	10.5	2.1	4.6	3.5	0.348
Deandre Ayton	Forward	PHO	18.2	11.5	1.9	0.3	0.231
Marvin Bagley III	Forward	SAC	14.2	7.5	0.8	1.7	0.182
Lonzo Ball	Guard	NOP	11.8	6.1	7	6.3	0.375

... (163 rows omitted)

Review

Bar Charts

Bar charts are often used to display the **relationship** between a categorical variable and a numerical variable:

- Average GPAs of Data Science, History, and Biology majors.
- The number of streams by the top 10 songs on Spotify yesterday.

Grouped Bar Chart

If the table we call `barh` on has multiple numerical columns, it will draw bars for each of them, and each column will get its own color!

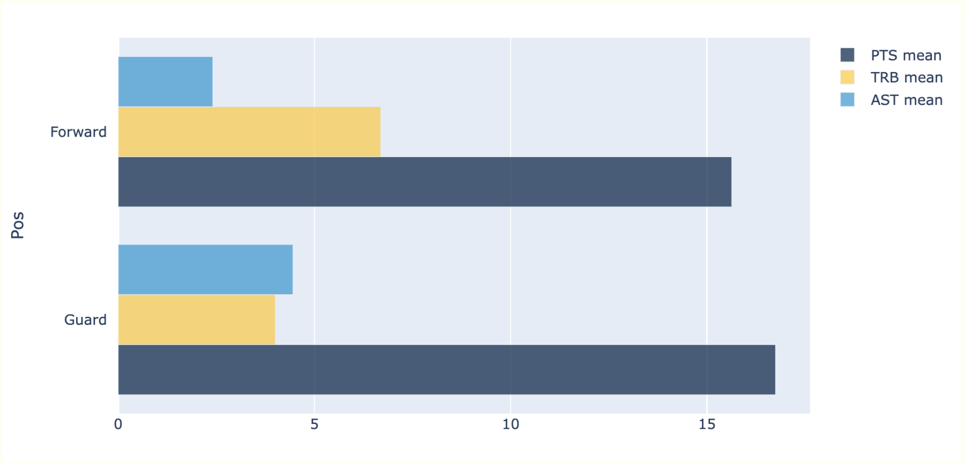
We create a Table that holds the mean number of points, rebounds and assists for the two positions (Forward and Guard). *Ignore the `.group()` method for now*

```
stats_by_pos = nba.group('Pos', np.mean).select('Pos', 'PTS mean', '
    TRB mean', 'AST mean')
stats_by_pos
```

Pos	PTS mean	TRB mean	AST mean
Forward	15.6297	6.68901	2.41099
Guard	16.7463	4.00244	4.45244

This data can be visualized with a grouped bar chart

```
stats_by_pos.barh('Pos')
```



Histograms

A histogram visualizes the distribution of a numerical variable by binning. The method

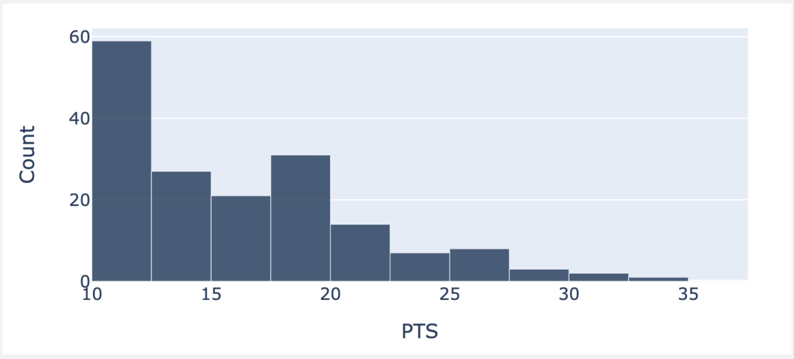
```
t.hist(column, density = False)
```

creates a histogram of the column column of `t`. This column must contain numerical values.

- It automatically chooses bins for us. We can change them.
- We will always set `density = False`.

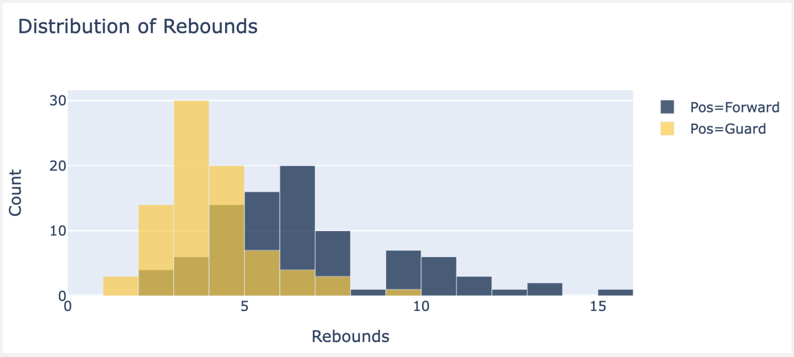
Create a histogram to visualize the distribution of points for the NBA players.

```
nba.hist('PTS', density = False, bins = np.arange(10, 40, 2.5))
```



Create a histogram to visualize the distributions of numerical variables (rebounds) by category (forward vs. guards).

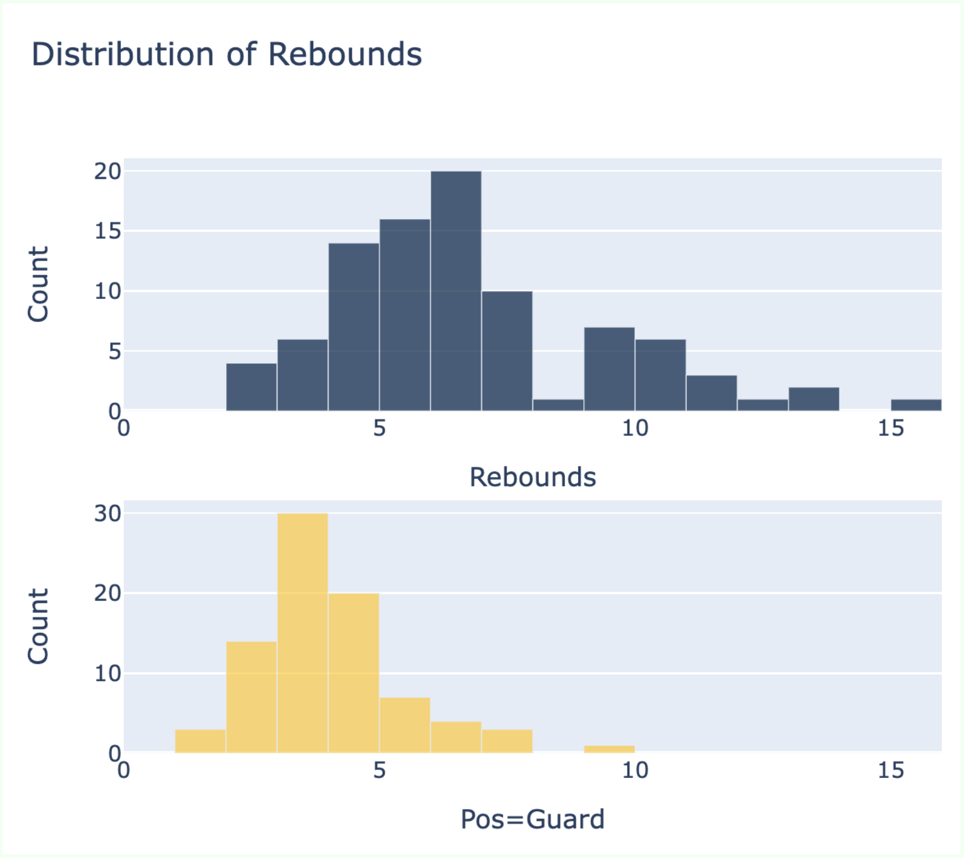
```
nba.hist('TRB', density = False, group = 'Pos', bins = np.arange(17),
         xaxis_title = 'Rebounds',
         title = 'Distribution of Rebounds')
```



How would you change the code to create two separate histograms one for forwards and one for guards?

Add an argument to the `.hist` method with `overlay = False`

```
nba.hist('TRB', density = False, group = 'Pos', bins = np.arange(17),
         xaxis_title = 'Rebounds',
         overlay = False,
         title = 'Distribution of Rebounds')
```



Scatter Plots

So far, we've visualized the following **combinations** of variables:

- **Bar Chart**: One categorical variable, one numerical variable
  - Top songs on Spotify
- **Histogram**: One numerical variable
  - Distribution of tips
  - Frequency of cookies

What if we want to visualize two numerical variables at once?

- Height vs. weight
- Tip vs. total bill
- Number of rebounds vs. number of points

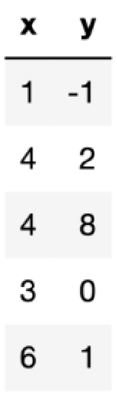
Scatter Plots

Scatter plots are used to visualize two numerical variables at once. To create a scatter plot from a table, you need two columns:

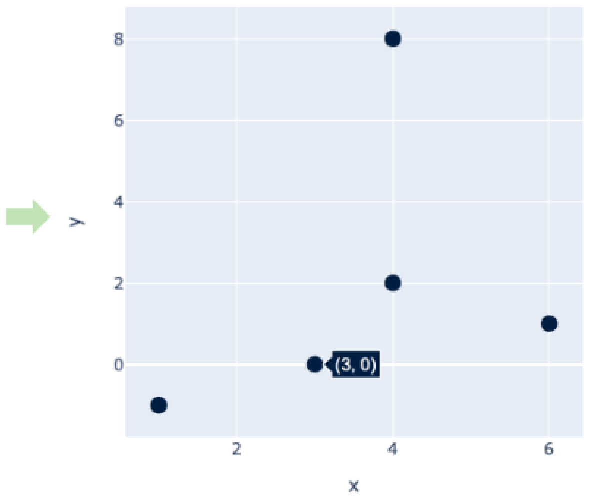
- A numerical column for the x-axis.
- A numerical column for the y-axis.

The resulting graph has one point for every row in your table.

- **We call this a graph of "y vs. x".**



x	y
1	-1
4	2
4	8
3	0
6	1



Create example data for plotting:

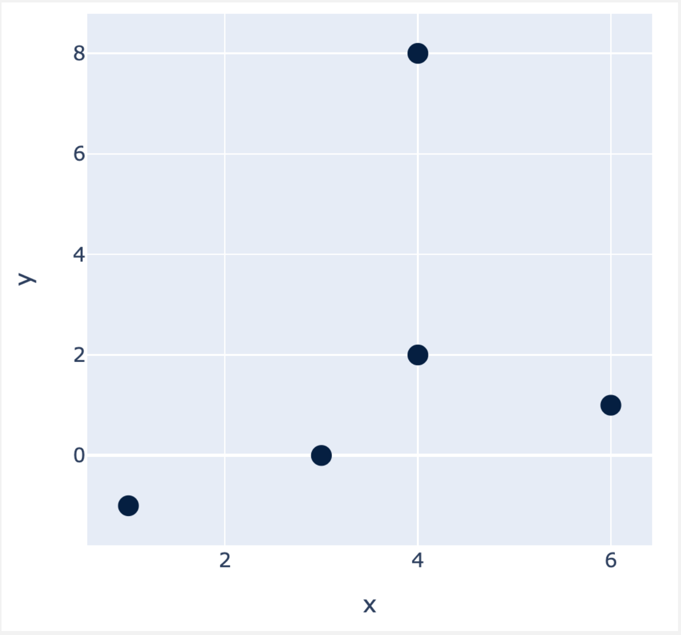
```
example_data = Table().with_columns(
    'x', np.array([1, 4, 4, 3, 6]),
    'y', np.array([-1, 2, 8, 0, 1])
)

example_data
```

x	y
1	-1
4	2
4	8
3	0
6	1

Create the scatter plot:

```
example_data.scatter('x', 'y', s = 50, width = 500, height = 500)
```



`.scatter()`

The method

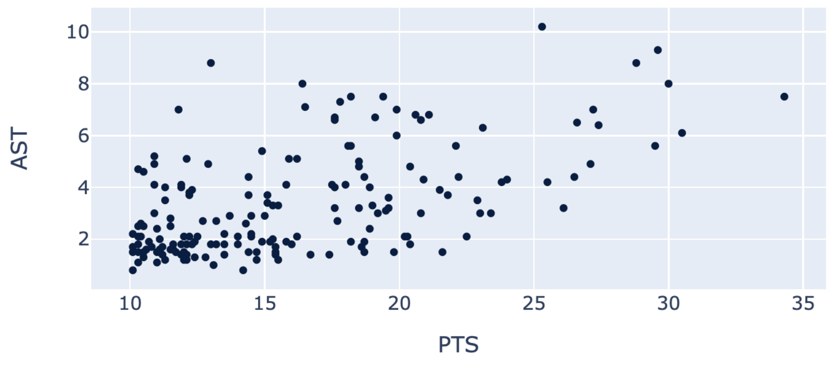
`t.scatter(column_for_x, column_for_y)`

creates a scatter plot using the specified columns. Both columns must contain numerical values.

- If only `column_for_x` is provided, a separate scatter plot is drawn for every other column in `t` (similar to the behavior of `barh`).

Example 1

```
nba.scatter('PTS', 'AST')
```



Observation

As the number of points a player averages increases, the number of assists they average also increases.

Quick Check 1

Fill in the blanks to create the scatter plot showing three point attempts (`'3PA'`) vs. rebounds (`'TRB'`) for forwards.

`nba.where(..., ...).scatter(..., ...)`

```
nba.where('Pos', 'Forward') \
          .scatter('TRB', '3PA',
           xaxis_title = 'Rebounds Per Game (TRB)',
           yaxis_title = 'Three-Point Attempts Per Game (3PA)',
           title = '3PA vs. TRB for Forwards')
```

Confirm with your neighbors

Then, check in the notebook

Customizing Scatter Plots

Customize `.barh`, `.hist`, `.scatter`

Along with `barh` and `hist`, we can use `xaxis_title`, `yaxis_title`, `title`, `width`, and `height` to tweak `scatter` plots.

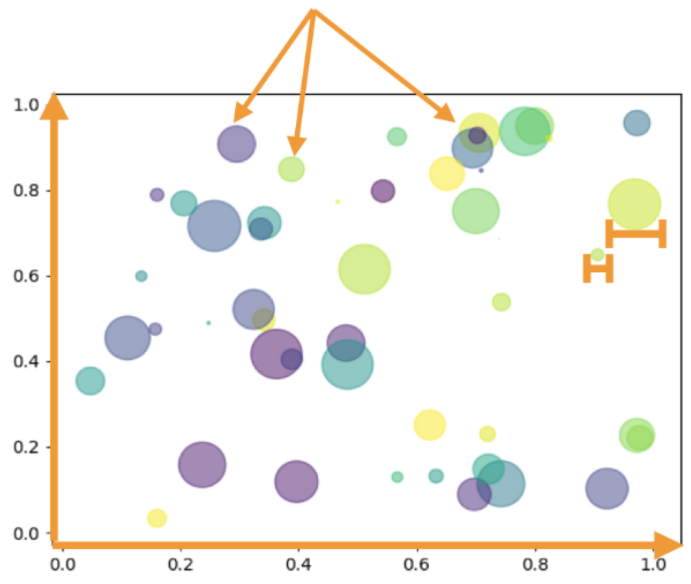
The optional arguments for `xaxis_title`, `yaxis_title`, and `title` will be set to `strings` that will be used as the x- and y-axis labels and title for the plots.

The optional arguments for `width` and `height` can be set to "numbers" indicating the width and height of the figure.

Customize `.scatter`

We can also take things a step further, by changing the following properties for each point:

- Size – make all points bigger, or make size proportional to some other numerical variable (e.g., older players have larger points).
- Color – different colors for different categories (e.g., one color for forwards, one color for guards).
- Labels – labeling each point according to a category (name, position, team, etc.).

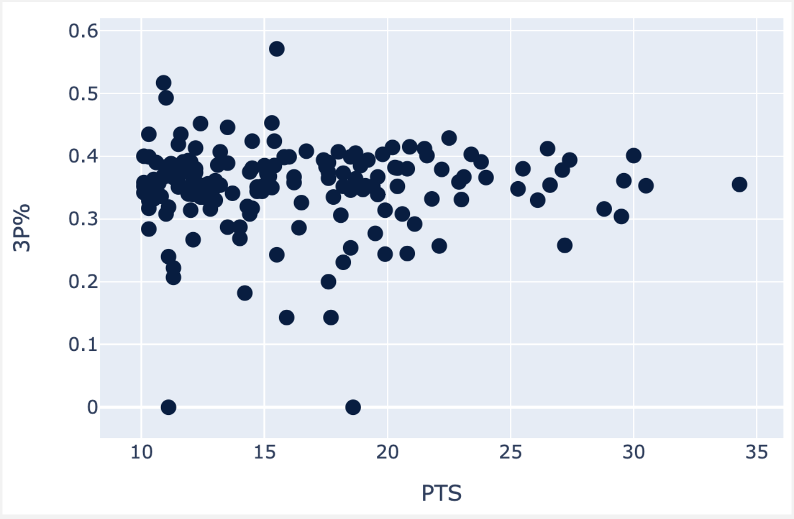


Point Size

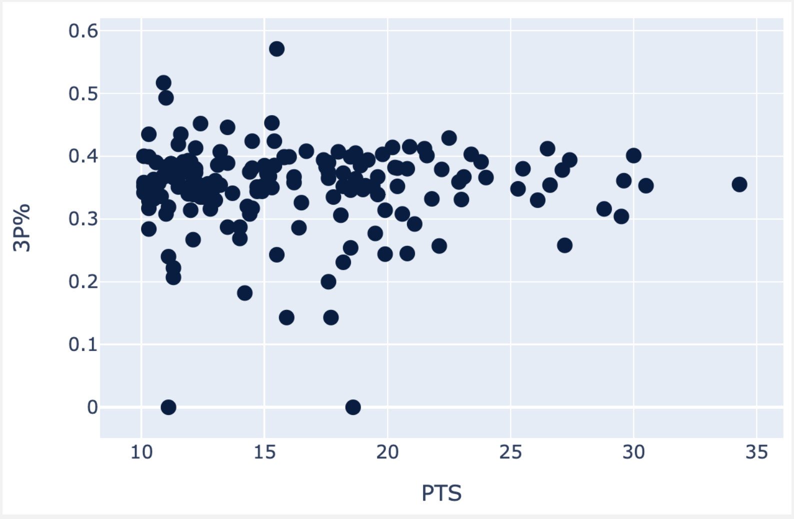
There are two relevant arguments:

- `s` (int): assign this to change the default size of all points.
- `sizes` (str): assign this to the name of a numerical column in your table; point sizes will be proportional to the values in this column.

```
nba.scatter('PTS', '3P%', s = 40)
```



```
nba.scatter('PTS', '3P%', s = 40, sizes = '3PA')
```



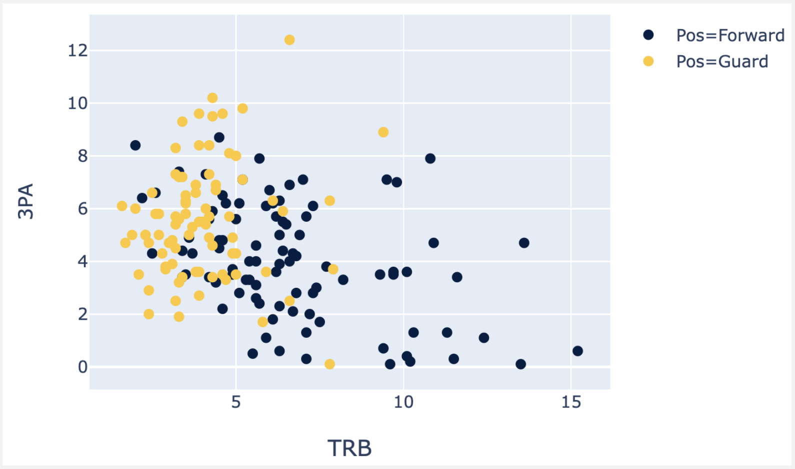
As 3PA increases, point size increases.

Point Color

`group` (str): assign this to the name of a categorical column in your table.

- Point colors will be determined according to the category.
- Effectively two separate scatter plots sharing the same axis.

```
nba.scatter('TRB', '3PA', group = 'Pos', s = 30)
```

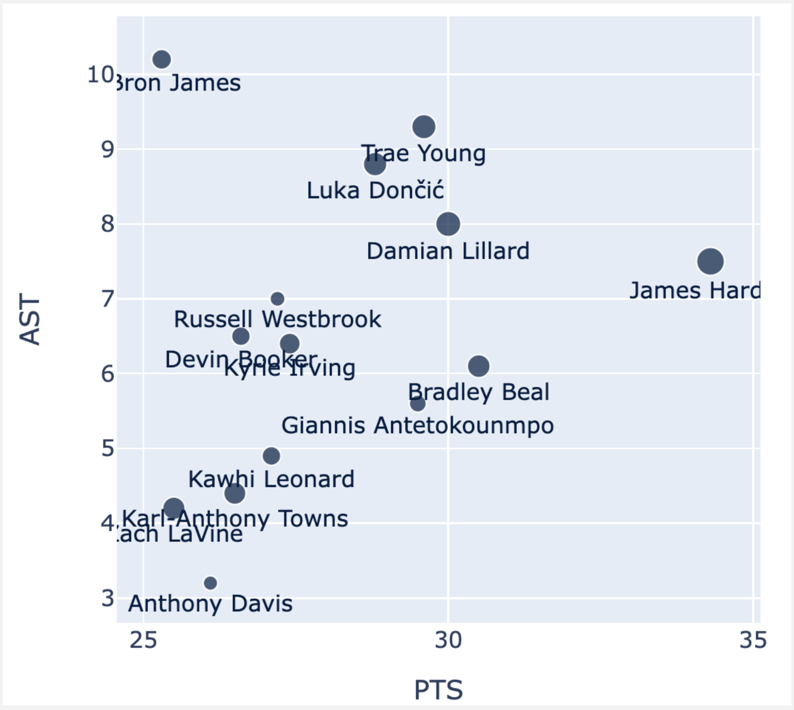


Guards tend to have fewer rebounds and more three-point attempts than forwards.

Labels

- `labels` (str): assign this to the name of any column in your table.
  - Each point will be labeled according to its value.
  - The more points you have, the harder the labels will be to read.
- You cannot combine **labels** and **group**.
  - Not currently implemented in `datascience`.
  - Can combine **labels** and **sizes**, and most other encodings.

```
nba.where('PTS', are.above(25)) \
   .scatter('PTS', 'AST',
           labels = 'Player',
           s = 30,
           width = 500,
           height = 500)
```



Line Plots

Data

A second dataset also comes from Basketball Reference. This dataset contains **team-based** average statistics for each year.

A little bit about our new dataset:

- `'Season'`: the second calendar year for each season (e.g. `2018` refers to the 2017-18 season).
- `'FGA'`: the average number of field goal attempts (shot attempts) per game.
- `'Pace'`: the average number of times a team had possession of the ball per game.

```
nba_yearly = Table.read_table('data/nba-league-averages.csv') \
                  .select('Season', 'PTS', 'FGA', '3PA', '3P%', 'Pace
                     ')
nba_yearly = nba_yearly.with_columns('Season', np.arange(2021, 1979,
    -1))
nba_yearly
```

Season	PTS	FGA	3PA	3P%	Pace
2021	111.7	88.3	34.7	0.367	99.2
2020	111.8	88.8	34.1	0.358	100.3
2019	111.2	89.2	32	0.355	100
2018	106.3	86.1	29	0.362	97.3
2017	105.6	85.4	27	0.358	96.4
2016	102.7	84.6	24.1	0.354	95.8
2015	100	83.6	22.4	0.35	93.9
2014	101	83	21.5	0.36	93.9
2013	98.1	82	20	0.359	92
2012	96.3	81.4	18.4	0.349	91.3

... (32 rows omitted)

Motivating Line Plots

What if we want to visualize two numerical variables, but one of them is time?

- COVID cases per day in Alameda County.
- Average rainfall for each month of the year in San Diego.

While a scatter plot would theoretically work in such a scenario, there are some key differences that lead us to another type of plot.

- There's only **one y for every x**.
  - There's only one number of COVID cases per day.
  - There can be many people with the same height when graphing weight vs. height.
- We want to emphasize a trend by "connecting the dots".

`.plot()`

The method

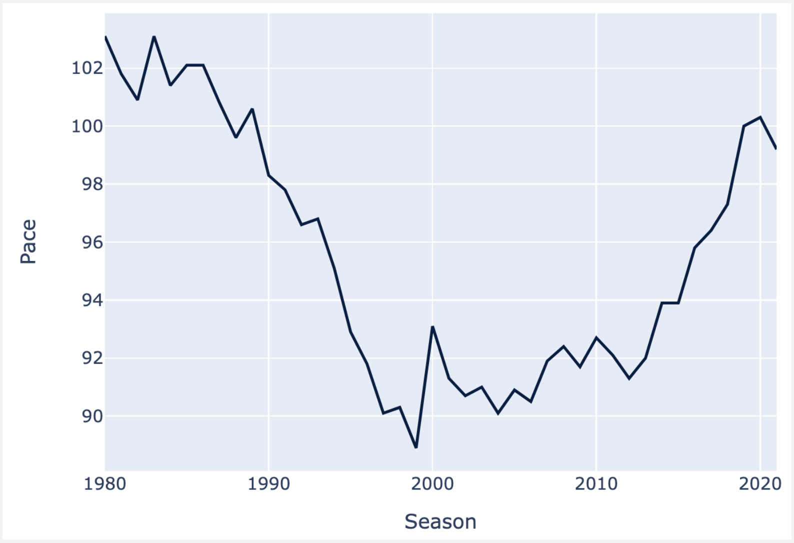
`t.plot(column_for_x, column_for_y)`

creates a line plot using the specified columns. Both columns must contain numerical values.

- `column_for_x` should contain some time-based variable.
- If only `column_for_x` is provided, a separate line plot is drawn for every other column in `t` (similar to the behavior of `barh` and `scatter`).

Example 1

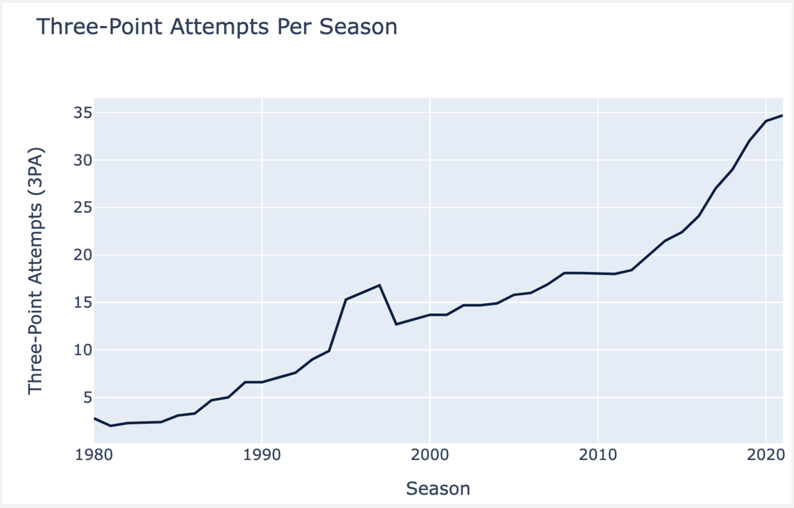
```
nba_yearly.plot('Season', 'Pace')
```



Observation: The league slowed down in the late 90s and early 2000s, but is speeding back up.

Example 2

```
nba_yearly.plot('Season', 'Pace')
```



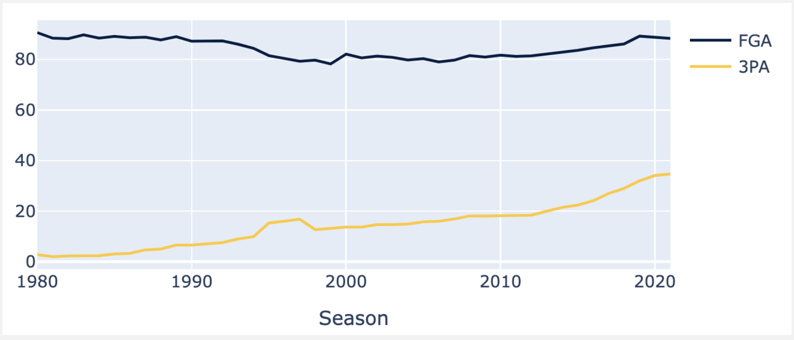
Observation: The three-point shot has rapidly increased in popularity over the past decade.

Multiple Line Plots

If we provide plot with only a single column name, it will draw lines for every other column in your table.

If you want to do this, make sure to select columns first!

```
nba_yearly.select('Season', 'FGA', '3PA').plot('Season')
```



Observation: Three point attempts have increase a lot since the 1980s, while the number of field goals (shots) attempted has stayed more or less the same.

Conclusion

Scatter plots visualize the relationship between any two numerical variables.

- No need to have unique x (or y) values.
- Useful for identifying patterns between variables

Line plots visualize the relationship between two numerical variables — one of them is ordered.

- x-axis generally represents time or distance.
- There should only be one y value for every x value.
- Useful for identifying trends over time

`scatter`

The method

`t.scatter(column_for_x, column_for_y)`

creates a scatter plot using the specified columns. Both columns must contain numerical values.

Optional arguments, in addition to `xaxis_title`, `width`, etc:

- `s` (int): changes default size of all points.
- `sizes` (str): point sizes will be proportional to the values in this numerical column.
- `group` (str): points will be colored according to category in this categorical column.
- `labels` (str): points will be labeled according to their value in this column.

`plot`

The method

`t.plot(column_for_x, column_for_y)`

creates a line plot using the specified columns. Both columns must contain numerical values.

- `column_for_x` should contain some time-based variable.
- If only `column_for_x` is provided, a separate line plot is drawn for every other column in `t` (similar to the behavior of `barh` and `scatter`).

Optional arguments, in addition to `xaxis_title`, `width`, etc:

- `overlay` (bool): If drawing multiple lines, setting `overlay` to `False` will draw multiple separate plots.

For the remainder of lab today, get practive with using the Table and Visualization commands from the last few weeks in the Lab 12 Worksheet.