DANIELS
COLLEGE OF BUSINESS
UNIVERSITY of DENVER

Python for Business Analytics – INFO 4120

Instructor: Neba Nfonsang

Module 3: STATISTICS

Objectives: Lesson 1

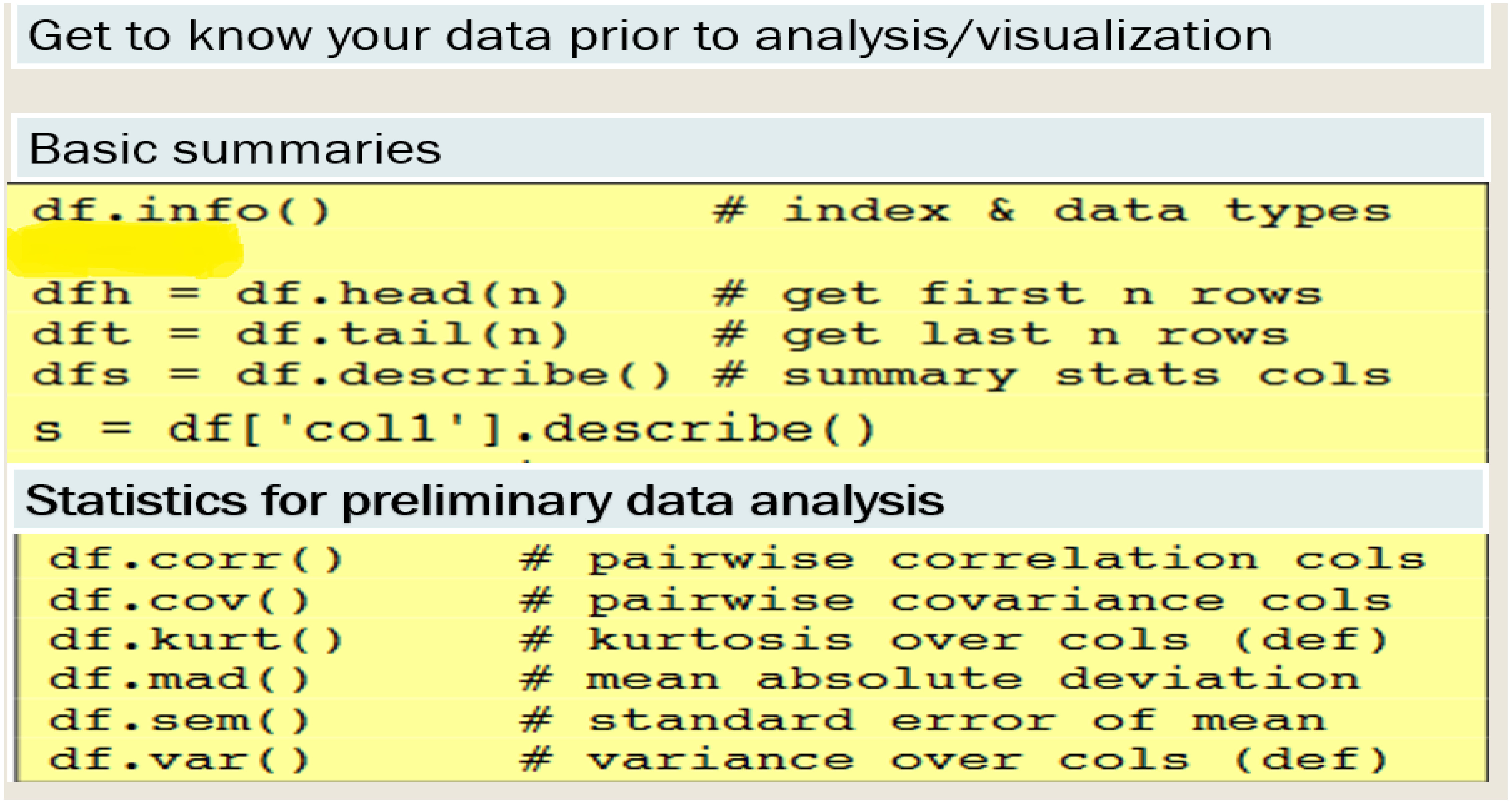
Statistical Analysis (STAT)

- Basic Summaries
  - Quantitative Data
    - Summarizing and Computing Descriptive Statistics and Histograms
  - Qualitative Data
    - Unique Values, Value Counts, CrossTabs, and Bar Plots
- Graphing with `matplotlib.pyplot`
  - Histogram
  - Bar/Column Chart, Pie Charts
  - Line Plots

Getting Started

- Lets create a DataFrame containing scores of 10 students in Statistics, Marketing, and Analytics.
- Students are either online and on campus (student status)
- Use the np array or list to create this data. Then use the zip function to store the data into a list called data. Create the DataFrame
- Add names of students as index to your df

Use these basic summaries with the table you created



Get to know your data prior to analysis/visualization

Basic summaries

```
df.info()                   # index & data types

dfh = df.head(n)            # get first n rows
dft = df.tail(n)            # get last n rows
dfs = df.describe() # summary stats cols
s = df['col1'].describe()
```

Statistics for preliminary data analysis

```
df.corr()           # pairwise correlation cols
df.cov()            # pairwise covariance cols
df.kurt()           # kurtosis over cols (def)
df.mad()            # mean absolute deviation
df.sem()            # standard error of mean
df.var()            # variance over cols (def)
```

Descriptive Statistics

- **df.mean()** to compute mean for each column
- **df.median()** to compute median for each column
- **df.max()** to compute maximum value
- **df.min()** to compute minimum value
- **df.std()** to compute standard deviation
- **df.corr()** to compute correlation
- **df.cov()** to compute covariance
- **df.kurt()** and **df.skew()** to compute kurtosis and skewness respectively

Parameter for some descriptive statistics

- Most descriptive statistics by default is performed across rows.
- If you want the statistics to be done across columns, you will need to pass the axis=1 or axis="columns" as parameter. For example:
- **df.mean(axis=1) or df.mean(axis="columns")**

Descriptive statistics for specific columns

- Generally use: **df.colname.statistic() or**
- **df[“colname”].statistics()**
- Examples:
  - **df.AnalyticsScore.mean() or**
  - **df[“AnalyticsScore”].mean()**
- To perform statistics for more than one column
  - **df[“AnalyticsScore”, “MarketingScore”].mean()**

Unique Values and Counts

- **df.colname.unique()** return the distinct levels of categorical data in the column
- **df.colname.value_counts()**
- **or pd.value_counts(df.colname)**
  - returns the distinct values (or categorical levels) with their frequencies or counts.
- Go ahead and try this on your table with the student status column (online or on campus)

Add another Column to your DataFrame so we can do crosstab

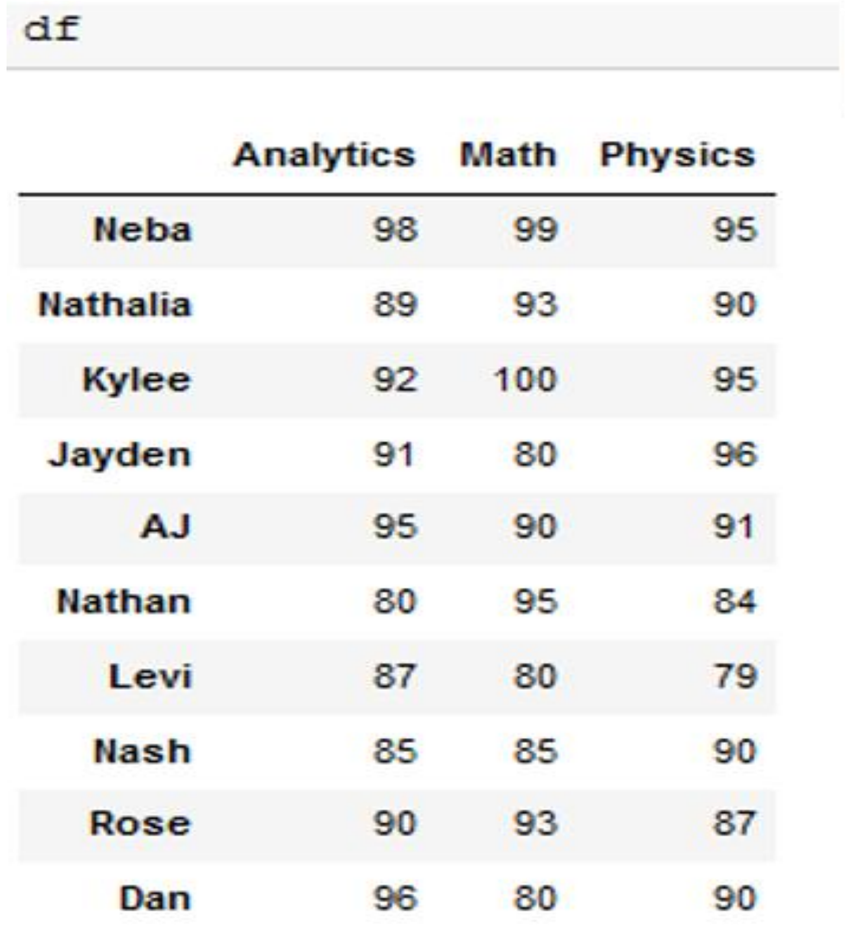
- Suppose all the students are coming from either Colorado, Nebraska or Utah.
- Create another column called “State” and randomly include the states for each student.
- There should be at least two students from each State (Colorado, Nebraska and Utah) .

CrossTabs

- Use the CrossTabs function to find how many online and on-campus students are from Colorado, Nebraska or Utah.
- Use the following code for cross tabulation
- **Status_State=pd.crosstab(df.Status, df.State)**
  - We can pass **margins=True** to output subtotals. We Can also pass **values=df.colname**, and **aggfunc=np.average** to calculate average using values of df.colname for the crosstab table. Also pass normalize=True to obtain data as fraction (percentages)
  - Note that the crosstab function returns a DataFrame.
- We can then use the crosstab table to generate bar charts or pie charts. Use **df.columns = ["newcolname"]** to rename the column and graph.

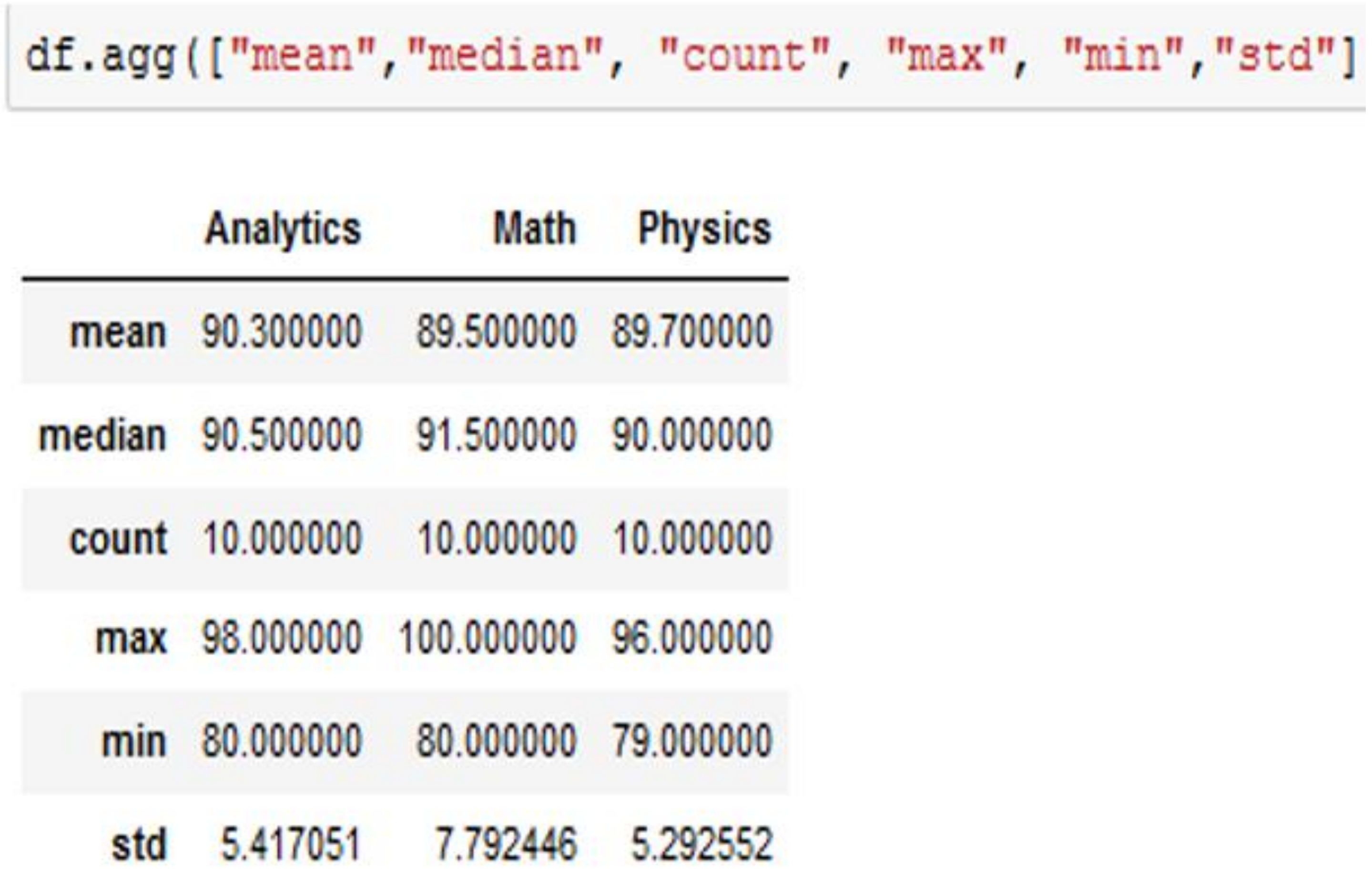
Aggregating and Grouping

We can use the df.agg() function and pass as parameters the mean, median, count, max, min, etc. to output these specific statistics.



```
df
```

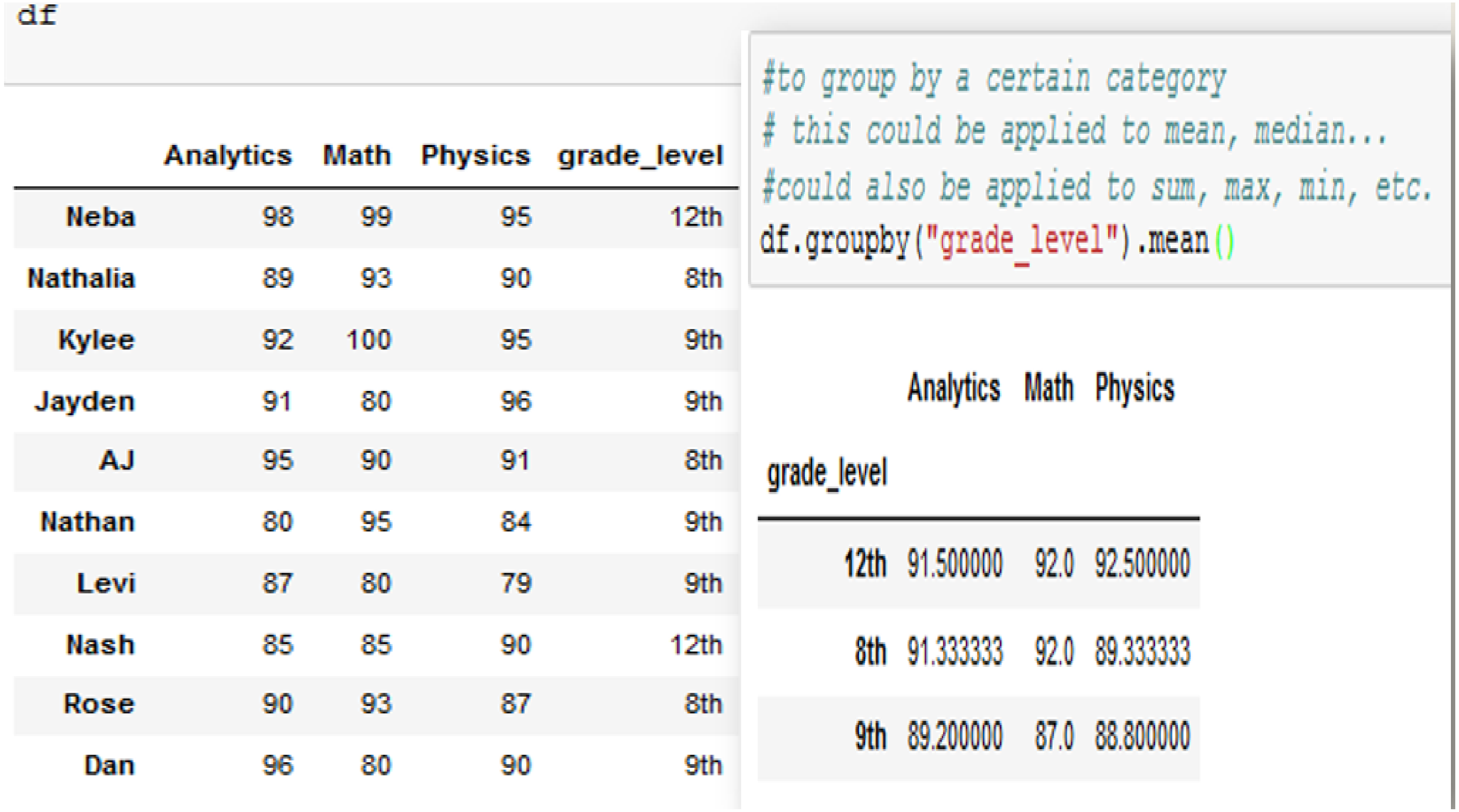
	Analytics	Math	Physics
Neba	98	99	95
Nathalia	89	93	90
Kylee	92	100	95
Jayden	91	80	96
AJ	95	90	91
Nathan	80	95	84
Levi	87	80	79
Nash	85	85	90
Rose	90	93	87
Dan	96	80	90



```
df.agg(["mean","median", "count", "max", "min","std"]
```

	Analytics	Math	Physics
mean	90.300000	89.500000	89.700000
median	90.500000	91.500000	90.000000
count	10.000000	10.000000	10.000000
max	98.000000	100.000000	96.000000
min	80.000000	80.000000	79.000000
std	5.417051	7.792446	5.292552

Grouping: df.groupby("catcolname").statistics



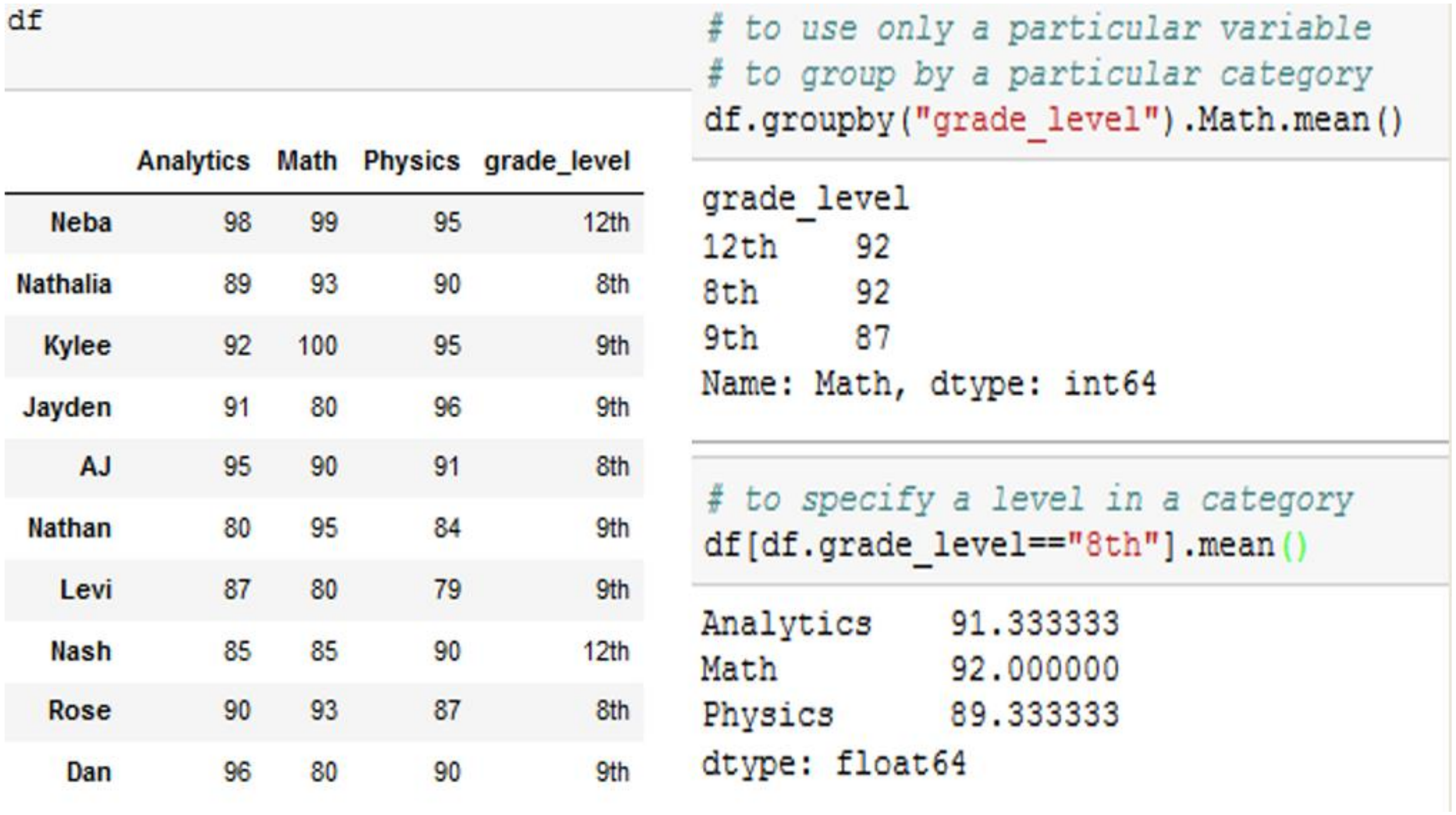
```
df
```

	Analytics	Math	Physics	grade_level
Neba	98	99	95	12th
Nathalia	89	93	90	8th
Kylee	92	100	95	9th
Jayden	91	80	96	9th
AJ	95	90	91	8th
Nathan	80	95	84	9th
Levi	87	80	79	9th
Nash	85	85	90	12th
Rose	90	93	87	8th
Dan	96	80	90	9th

```
#to group by a certain category
# this could be applied to mean, median...
#could also be applied to sum, max, min, etc.
df.groupby("grade_level").mean()
```

	Analytics	Math	Physics
grade_level			
12th	91.500000	92.0	92.500000
8th	91.333333	92.0	89.333333
9th	89.200000	87.0	88.800000

groupby() for specific columns



```
df
```

	Analytics	Math	Physics	grade_level
Neba	98	99	95	12th
Nathalia	89	93	90	8th
Kylee	92	100	95	9th
Jayden	91	80	96	9th
AJ	95	90	91	8th
Nathan	80	95	84	9th
Levi	87	80	79	9th
Nash	85	85	90	12th
Rose	90	93	87	8th
Dan	96	80	90	9th

```
# to use only a particular variable
# to group by a particular category
df.groupby("grade_level").Math.mean()
```

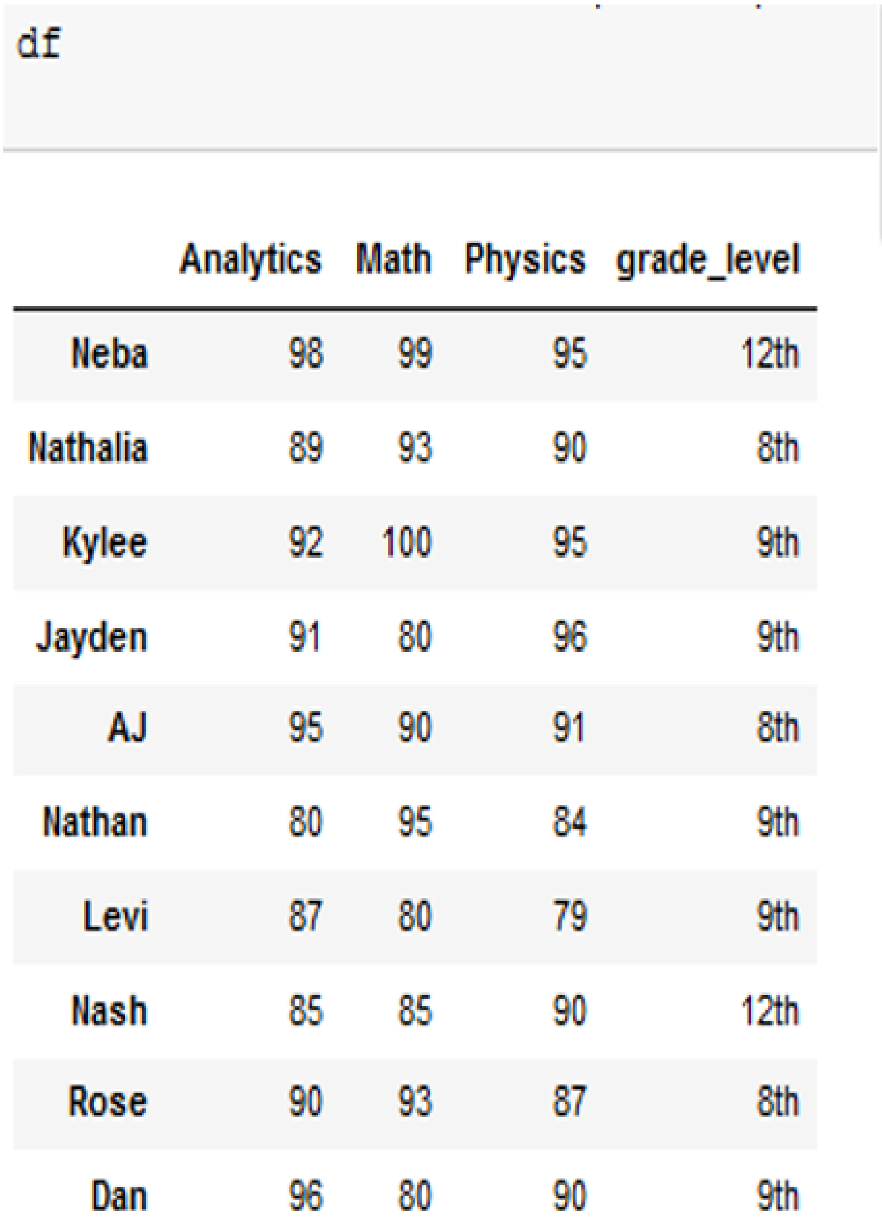
```
grade_level
12th     92
8th      92
9th      87
Name: Math, dtype: int64
```

```
# to specify a level in a category
df[df.grade_level=="8th"].mean()
```

```
Analytics    91.333333
Math         92.000000
Physics      89.333333
dtype: float64
```

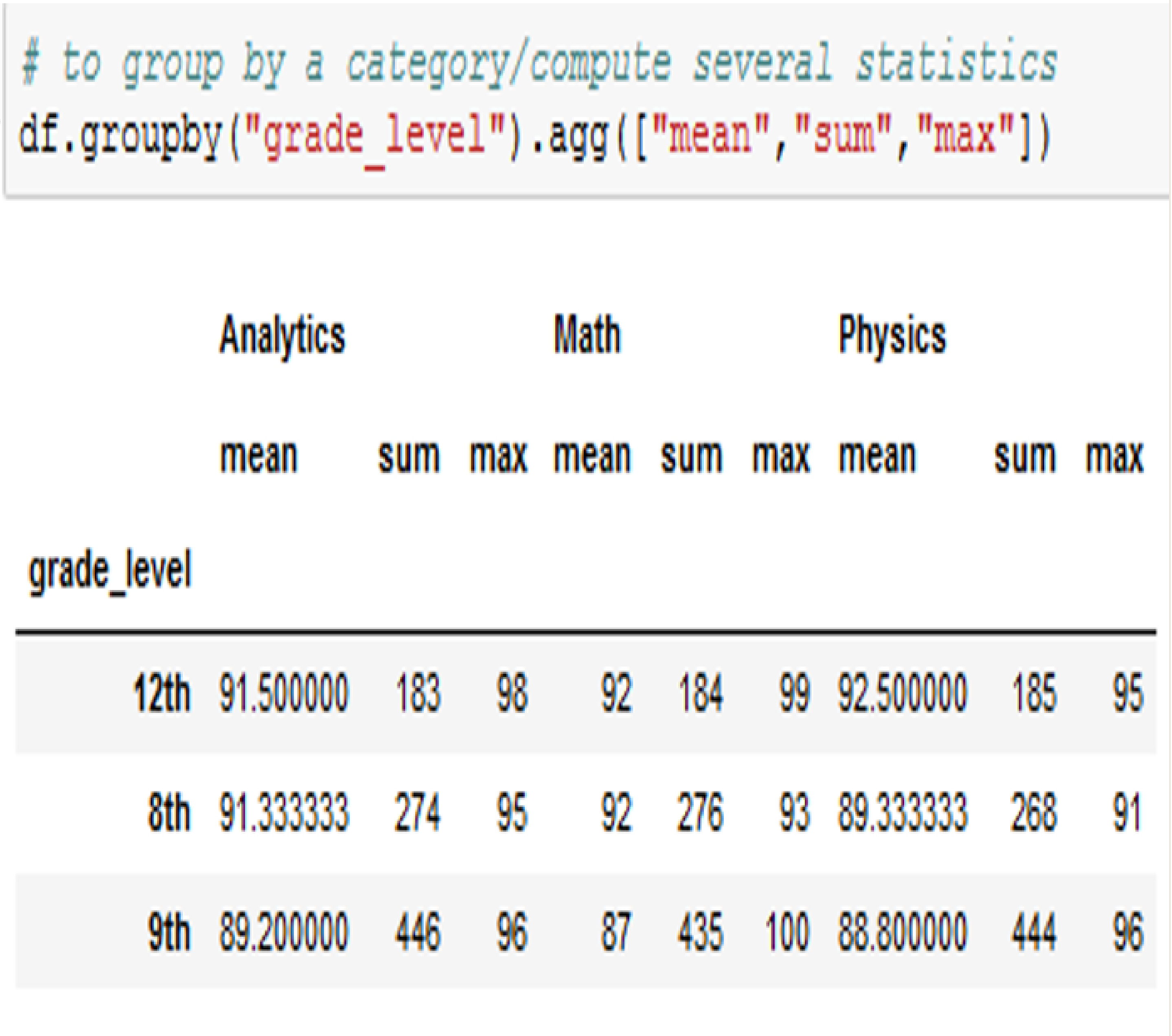
DANIELS
COLLEGE OF BUSINESS
UNIVERSITY of DENVER

Aggregating and Grouping Data



```
df
```

	Analytics	Math	Physics	grade_level
Neba	98	99	95	12th
Nathalia	89	93	90	8th
Kylee	92	100	95	9th
Jayden	91	80	96	9th
AJ	95	90	91	8th
Nathan	80	95	84	9th
Levi	87	80	79	9th
Nash	85	85	90	12th
Rose	90	93	87	8th
Dan	96	80	90	9th



```
# to group by a category/compute several statistics
df.groupby("grade_level").agg(["mean","sum","max"])
```

	Analytics			Math			Physics		
	mean	sum	max	mean	sum	max	mean	sum	max
grade_level									
12th	91.500000	183	98	92	184	99	92.500000	185	95
8th	91.333333	274	95	92	276	93	89.333333	268	91
9th	89.200000	446	96	87	435	100	88.800000	444	96

Plots in Pandas: df.plot.chartname()

- Use **df.colname.hist()** to plot histogram for a column
- Use **df[“col1”, “col2”].hist()** to select 2 columns etc

To plot for all columns in DataFrame

- df**.plot.hist() you can pass bins=20, color=**
- **df.plot.bar()**
- **df.plot.barh(Stacked=True)** to stack columns horizontally
- **df.plot.pie()**
- **df.plot.scatter(“col1”, “col2”)**
- **df.plot(kind="bar")** you can also pass “hist” as parameter
- Histogram is for continuous data while bar chart is for categorical data

Plot: Example 1

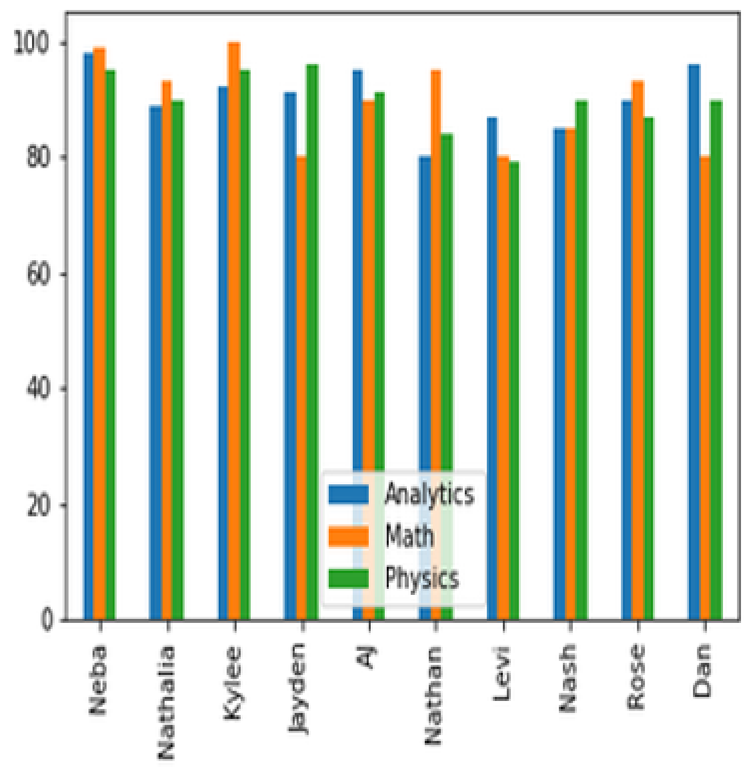
Visualization: Bar Chart

```
df
```

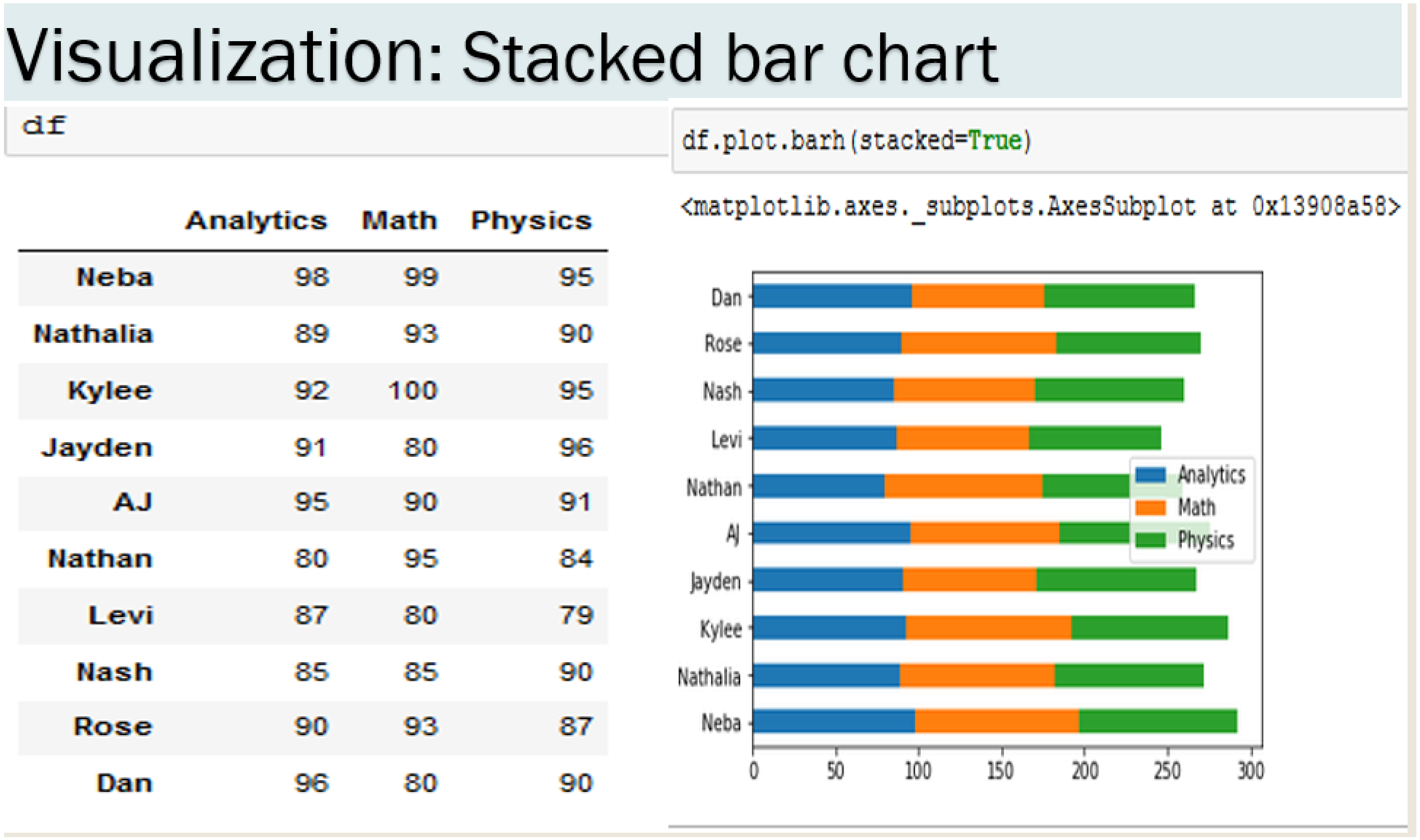
	Analytics	Math	Physics
Neba	98	99	95
Nathalia	89	93	90
Kylee	92	100	95
Jayden	91	80	96
AJ	95	90	91
Nathan	80	95	84
Levi	87	80	79
Nash	85	85	90
Rose	90	93	87
Dan	96	80	90

```
df.plot.bar()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x133daa58>
```



Plot: Example 2



Visualization: Stacked bar chart

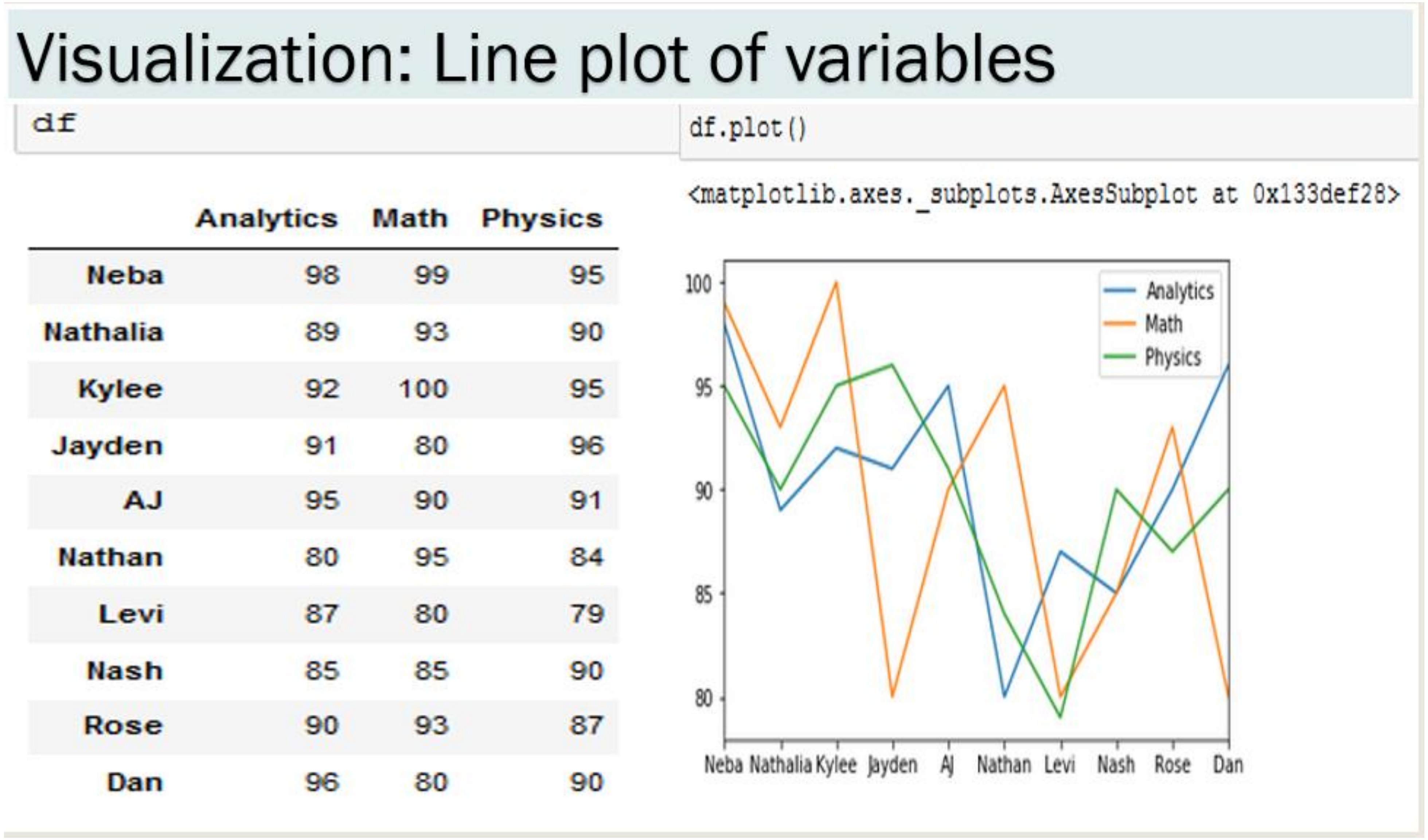
```
df
```

	Analytics	Math	Physics
Neba	98	99	95
Nathalia	89	93	90
Kylee	92	100	95
Jayden	91	80	96
AJ	95	90	91
Nathan	80	95	84
Levi	87	80	79
Nash	85	85	90
Rose	90	93	87
Dan	96	80	90

```
df.plot.barh(stacked=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x13908a58>
```

Plot: Example 3



Visualization: Line plot of variables

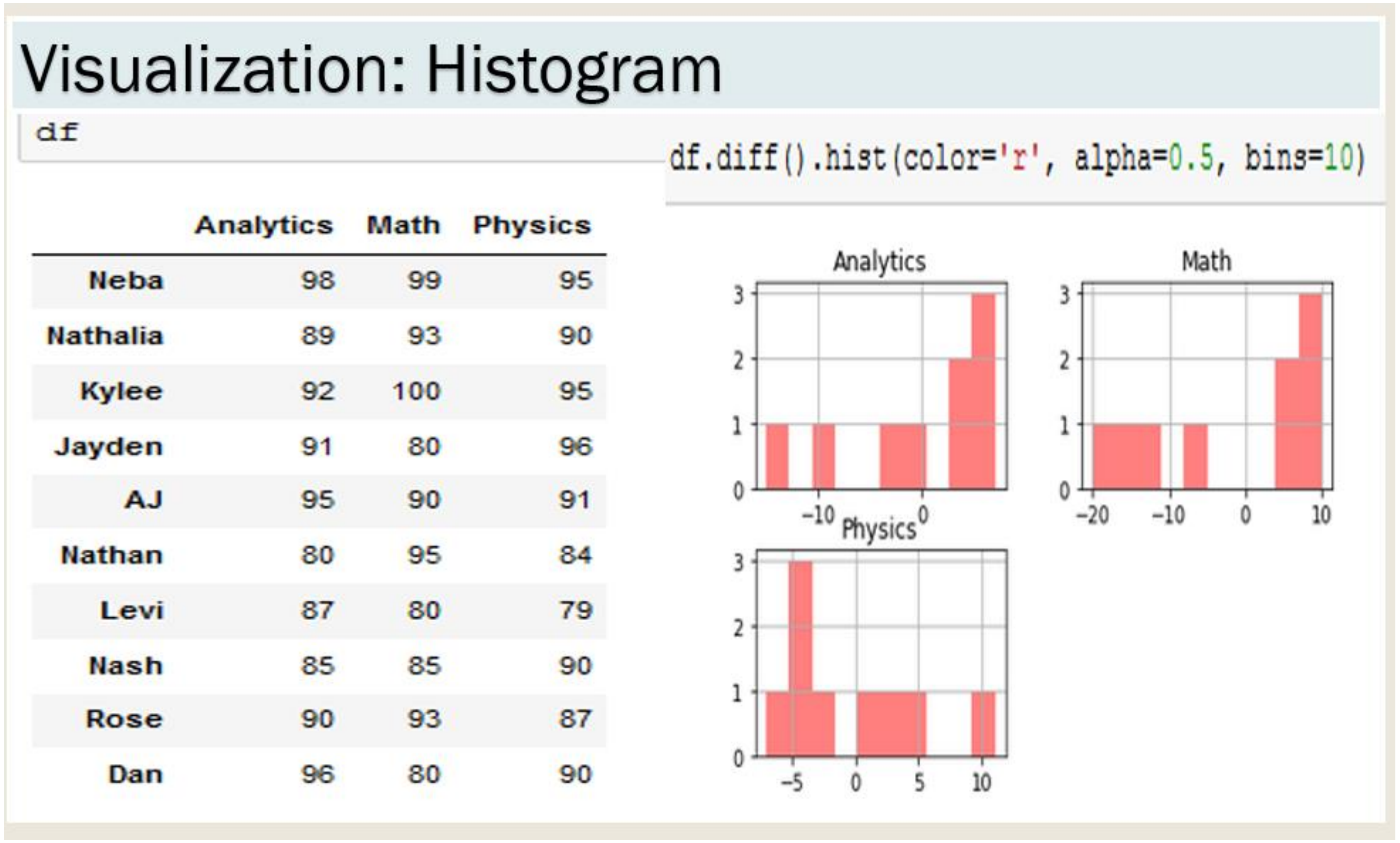
```
df
```

	Analytics	Math	Physics
Neba	98	99	95
Nathalia	89	93	90
Kylee	92	100	95
Jayden	91	80	96
AJ	95	90	91
Nathan	80	95	84
Levi	87	80	79
Nash	85	85	90
Rose	90	93	87
Dan	96	80	90

```
df.plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x133def28>
```

Plot: Example 4



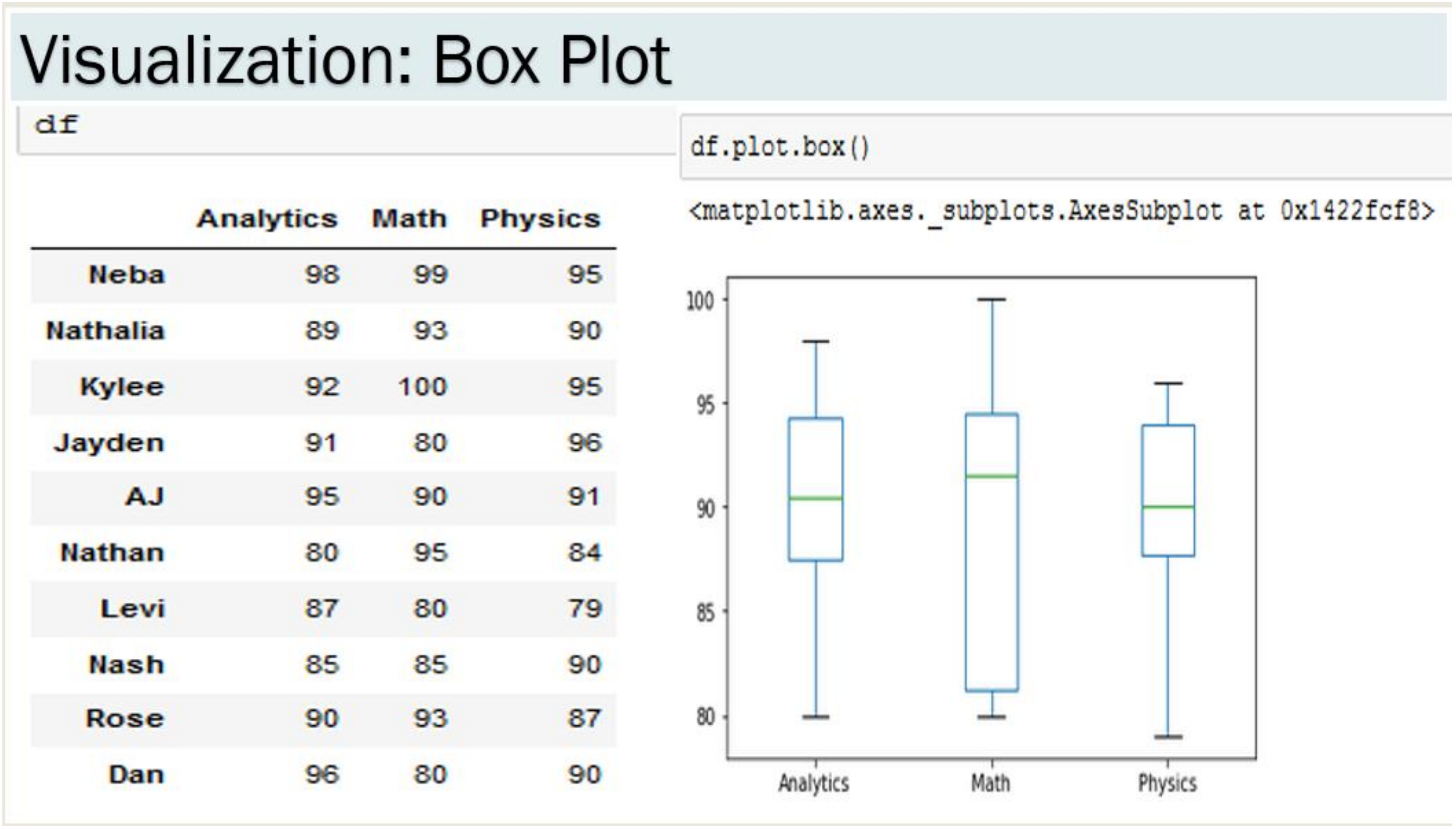
Visualization: Histogram

```
df
```

	Analytics	Math	Physics
Neba	98	99	95
Nathalia	89	93	90
Kylee	92	100	95
Jayden	91	80	96
AJ	95	90	91
Nathan	80	95	84
Levi	87	80	79
Nash	85	85	90
Rose	90	93	87
Dan	96	80	90

```
df.diff().hist(color='r', alpha=0.5, bins=10)
```

Plot: Example 5



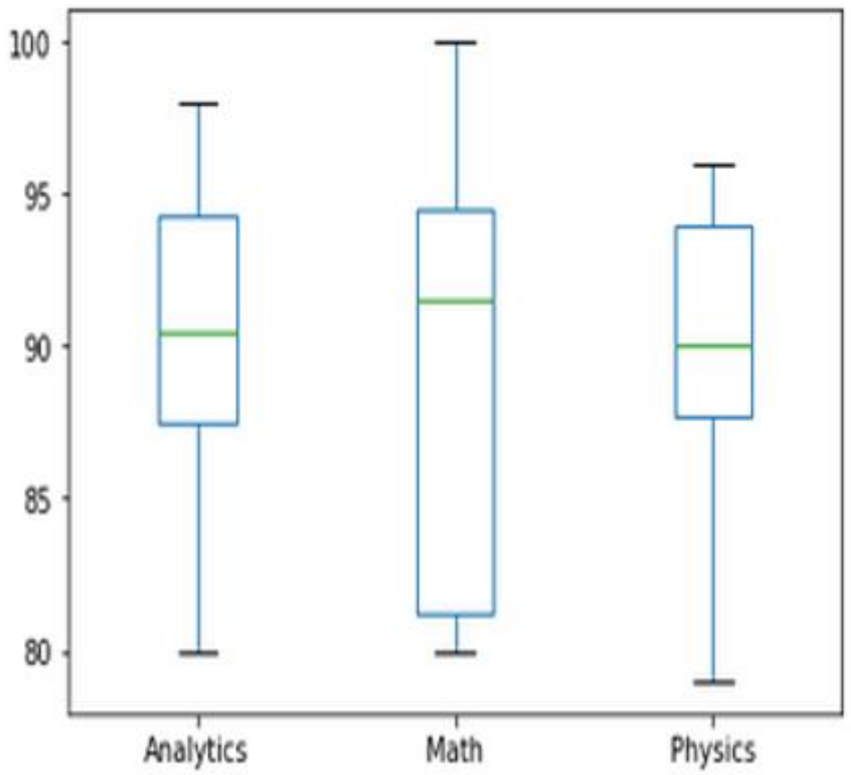
Visualization: Box Plot

```
df
```

	Analytics	Math	Physics
Neba	98	99	95
Nathalia	89	93	90
Kylee	92	100	95
Jayden	91	80	96
AJ	95	90	91
Nathan	80	95	84
Levi	87	80	79
Nash	85	85	90
Rose	90	93	87
Dan	96	80	90

```
df.plot.box()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1422fcf8>
```



Plot: Box plot example with parameters

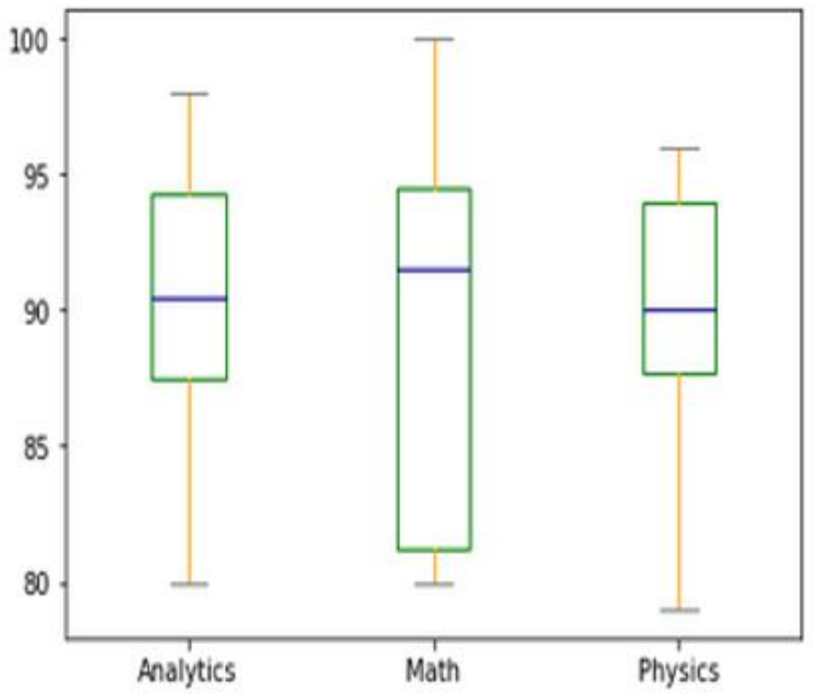
Visualization: Box Plot

```
df
```

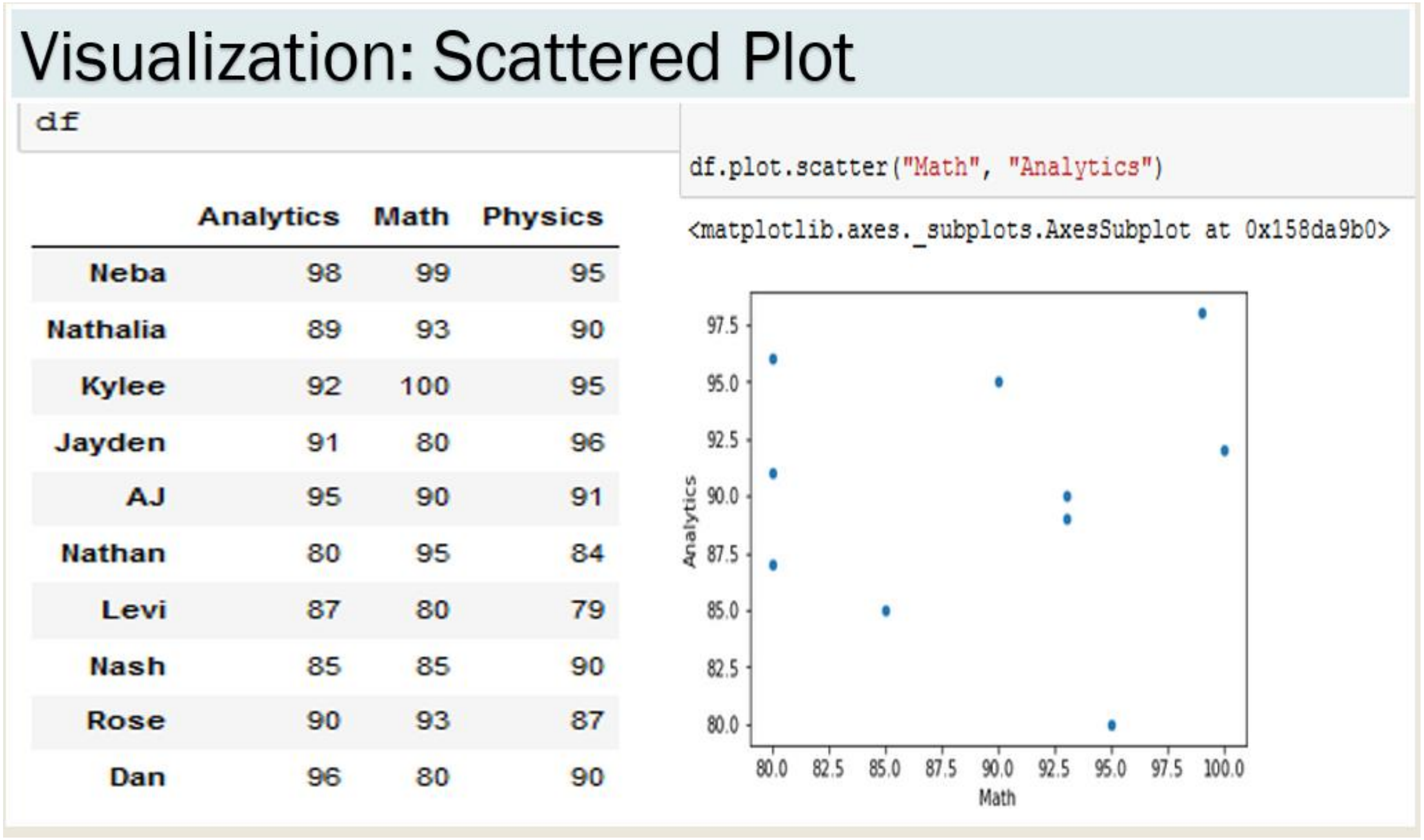
	Analytics	Math	Physics
Neba	98	99	95
Nathalia	89	93	90
Kylee	92	100	95
Jayden	91	80	96
AJ	95	90	91
Nathan	80	95	84
Levi	87	80	79
Nash	85	85	90
Rose	90	93	87
Dan	96	80	90

```
color = dict(boxes='Green', whiskers='Orange',
             medians='DarkBlue', caps='Gray')
df.plot.box(color=color, sym='r+')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x13bd44a8>
```



Plot: Example 6



Visualization: Scattered Plot

```
df
```

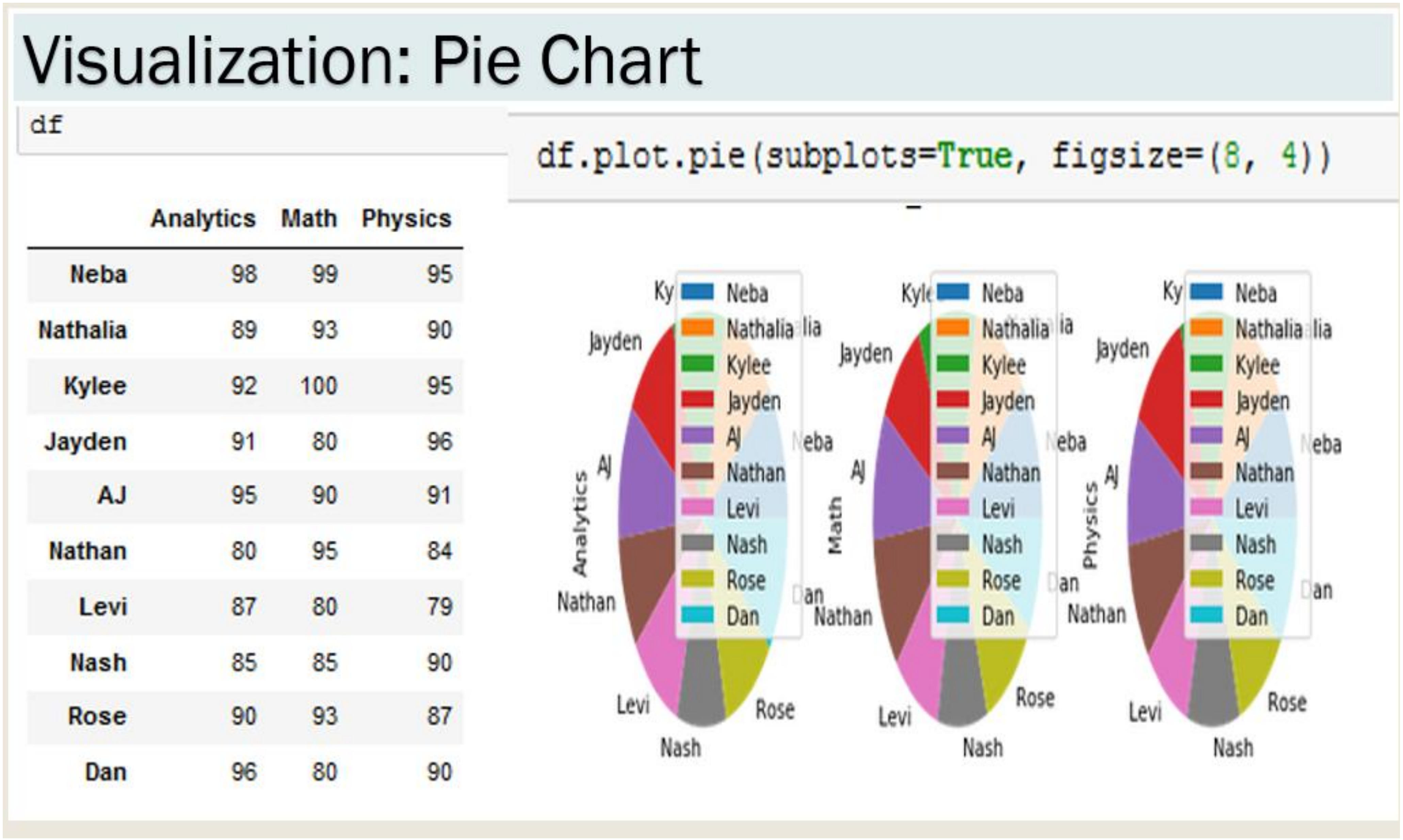
	Analytics	Math	Physics
Neba	98	99	95
Nathalia	89	93	90
Kylee	92	100	95
Jayden	91	80	96
AJ	95	90	91
Nathan	80	95	84
Levi	87	80	79
Nash	85	85	90
Rose	90	93	87
Dan	96	80	90

```
df.plot.scatter("Math", "Analytics")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x158da9b0>
```

DANIELS COLLEGE OF BUSINESS
UNIVERSITY of DENVER

Plot: Example 7

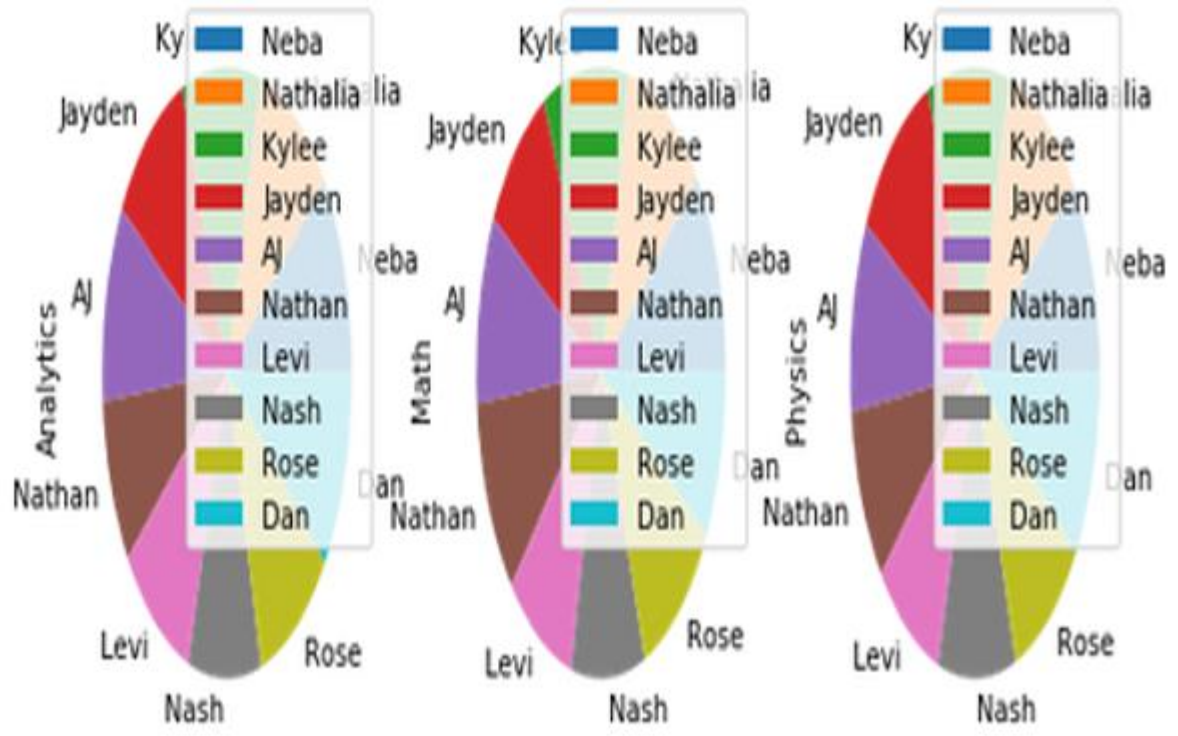


Visualization: Pie Chart

```
df
```

	Analytics	Math	Physics
Neba	98	99	95
Nathalia	89	93	90
Kylee	92	100	95
Jayden	91	80	96
AJ	95	90	91
Nathan	80	95	84
Levi	87	80	79
Nash	85	85	90
Rose	90	93	87
Dan	96	80	90

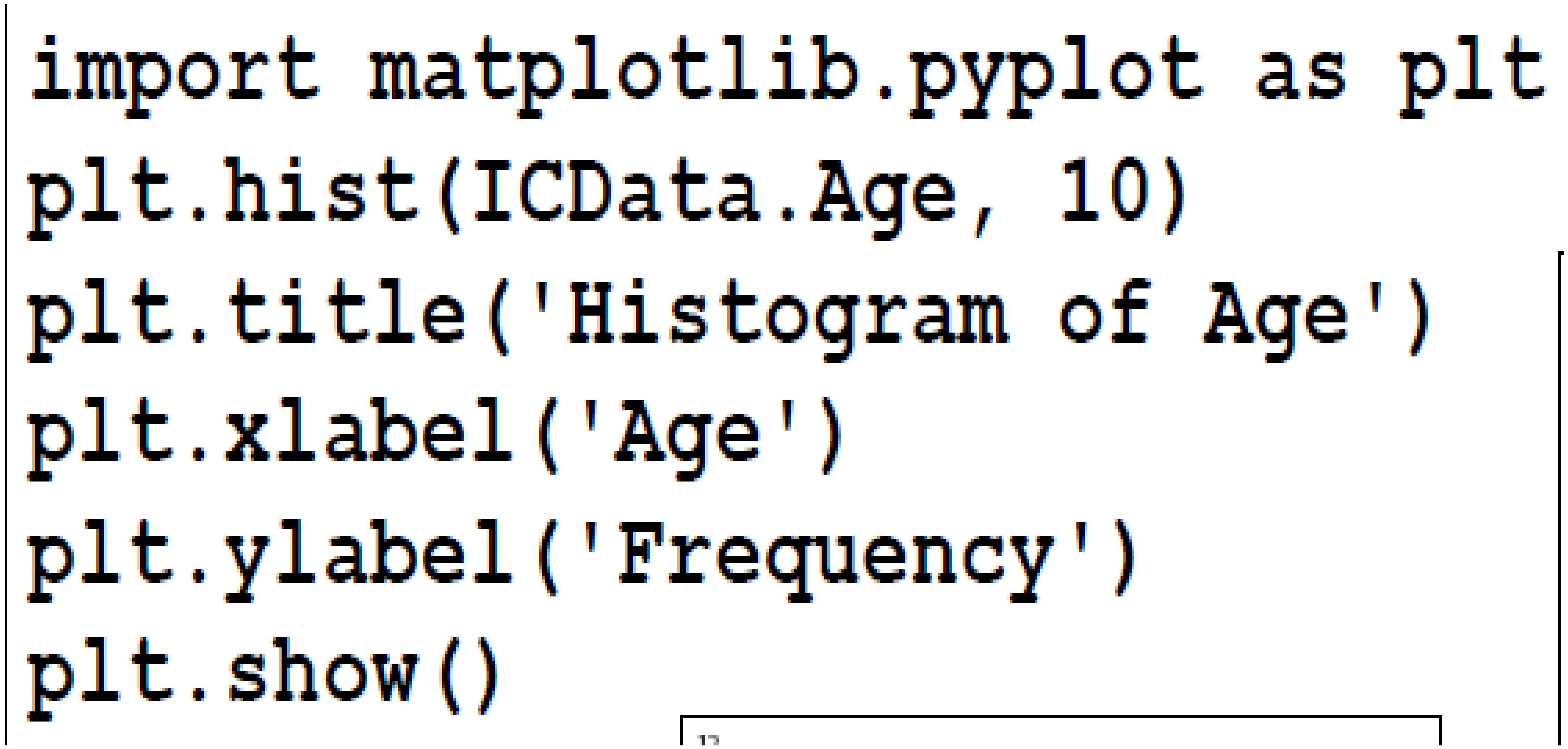
```
df.plot.pie(subplots=True, figsize=(8, 4))
```



Ploting with matplotblib.pyplot

```
import matplotblib.pyplot as plt
plt.hist(df.colname, bins=, )
```

Matplotlib.pyplot histogram



```
import matplotlib.pyplot as plt
plt.hist(ICData.Age, 10)
plt.title('Histogram of Age')
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.show()
```

Histogram and bar chart

- plt.hist(df.colname, bins= , range=(start, end), color=)
- To create a bar chart, first convert the categorical data into counts using the df.colname.value_counts() function
- Then store this value_counts() into an object (Series).
- Covert that object into a DataFrame
- From the data, use df.plot(kind="bar") to plot a Pandas bar chart.
- To plot a matplotblib bar chart, you will need to use
- plt.bar(x,y, width=0.5, color="green")
- Where x=categories, and y=number of counts or the count column in the count DataFrame.
- Y= df.countscolname
- X=np.arrange(len(y)) - to create numbers representing categories.

Pie Chart

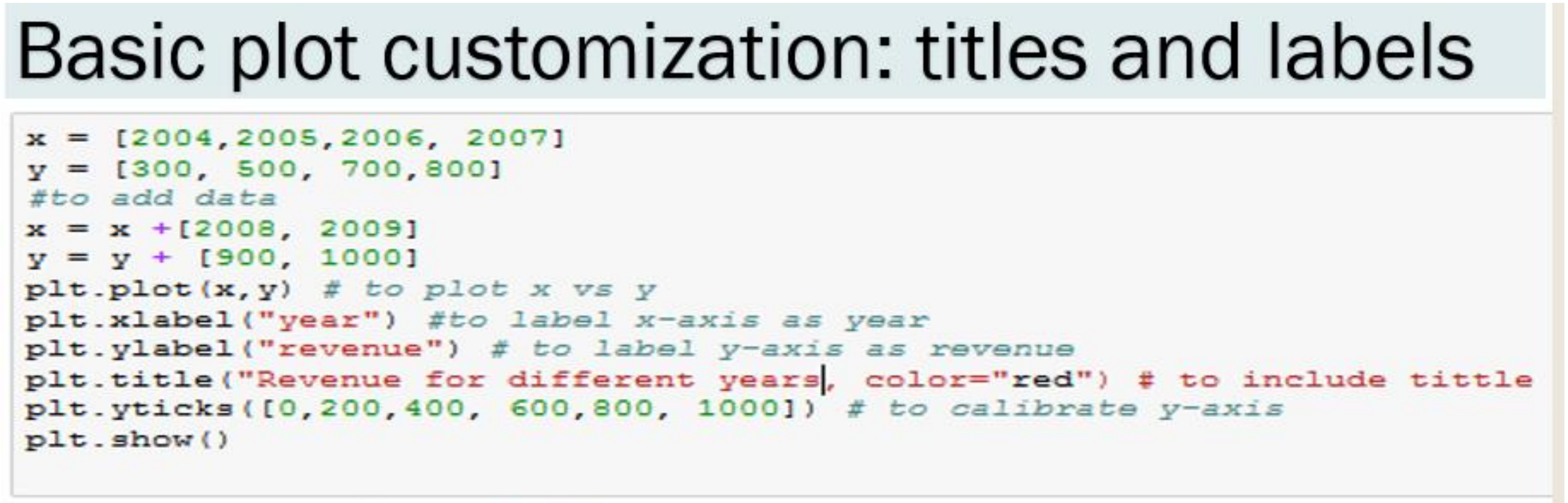
The syntax is: `plt.pie(data series,labels,colors)`

data = df.countcolumn,
labels= df.index
colors= [list of colors for number of catergories]

- Before you run the pie chart code:
- You need to use the **df.colunmName.value_count()**
- To count the column with categorical data,
- then store that data in an object
- Convert the object to a data frame. Change the column name of the DataFrame to count

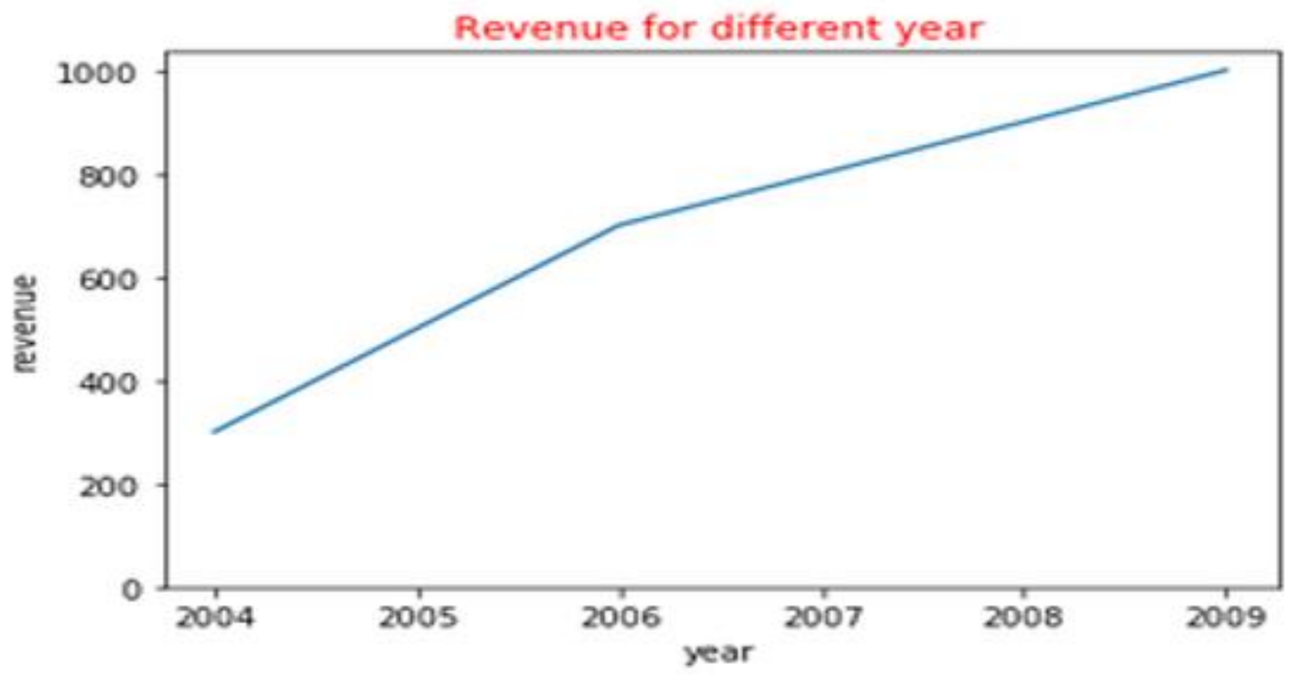
Use the shift and Tab key to see more parameters about these functions

Plotting a line graph example



Basic plot customization: titles and labels

```
x = [2004,2005,2006, 2007]
y = [300, 500, 700,800]
#to add data
x = x +[2008, 2009]
y = y + [900, 1000]
plt.plot(x,y) # to plot x vs y
plt.xlabel("year") #to label x-axis as year
plt.ylabel("revenue") # to label y-axis as revenue
plt.title("Revenue for different years", color="red") # to include tittle
plt.yticks([0,200,400, 600,800, 1000]) # to calibrate y-axis
plt.show()
```


Revenue for different year
revenue
year