Lecture 3

Visualization and Data Types

Dr. David Zmiaikou

Visualization

Two primary uses for data visualization:

- To explore data
- To communicate data

⏳ A bit of history…

- **John D. Hunter** (August 1, 1968 – August 28, 2012) was an American neurobiologist and the original author of Matplotlib.
- The project was started by John Hunter in 2002 to enable a MATLAB-like plotting interface in Python.

matplotlib.pyplot

We can use the matplotlib.pyplot module.

pyplot maintains an internal state in which you build up a visualization step by step.

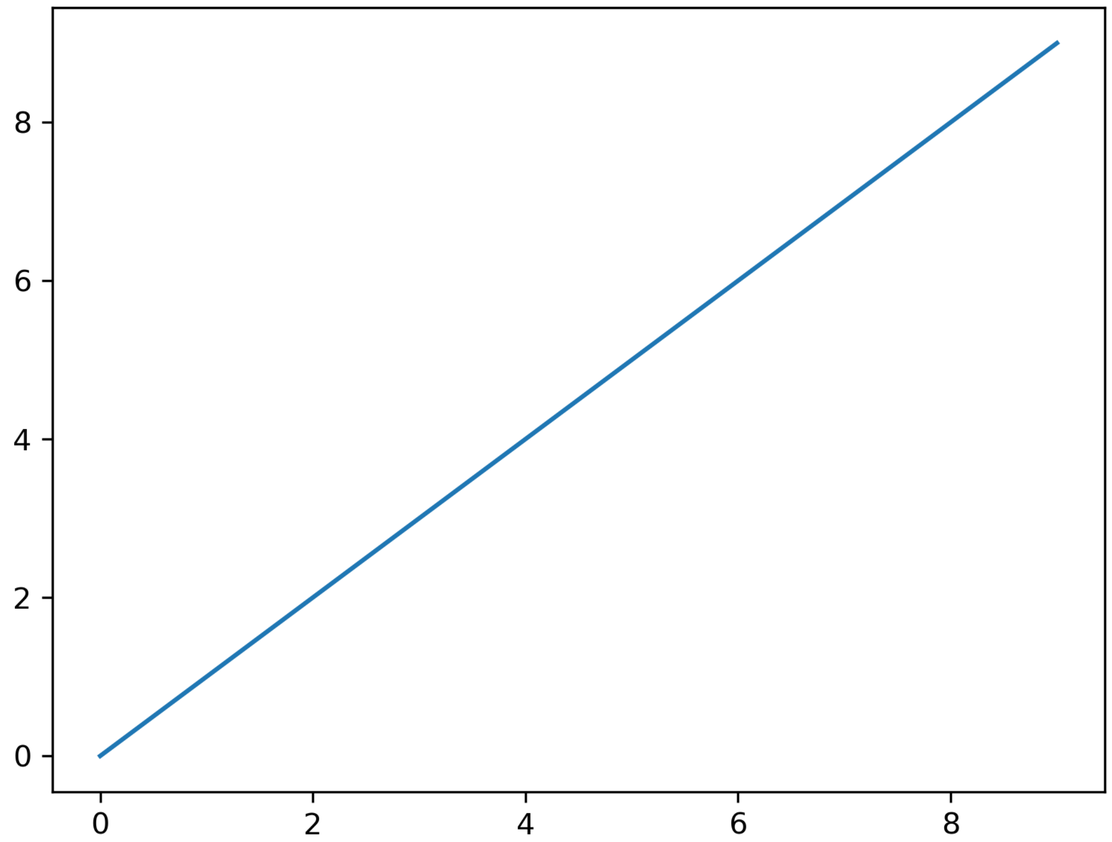
Once you're done, you can save it with **savefig** or display it with **show**.

Example 1

```
import matplotlib.pyplot as plt
import numpy as np

data = np.arange(10)
plt.plot(data)
```

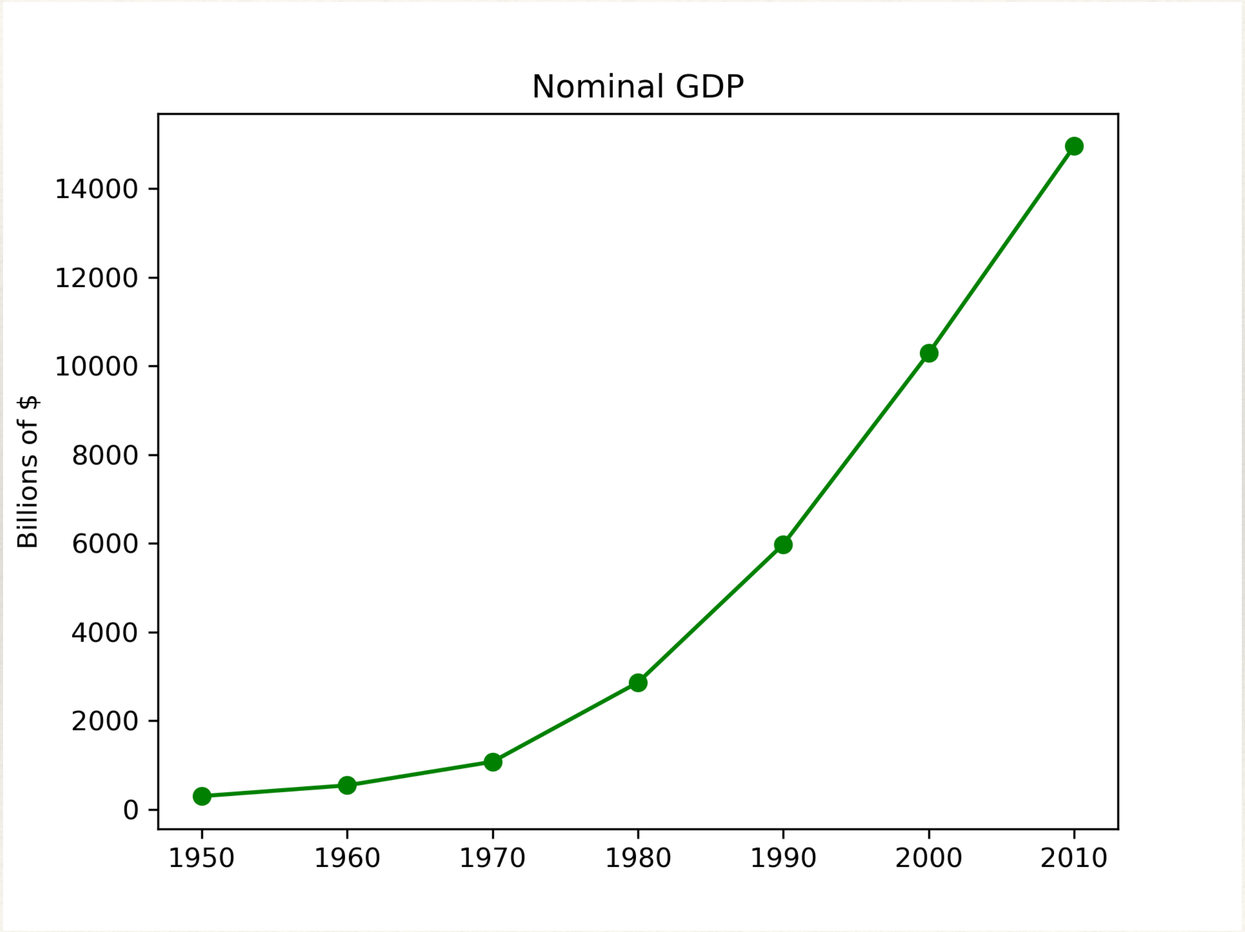
Example 1

```
import matplotlib.pyplot as plt
import numpy as np

data = np.arange(10)
plt.plot(data)
```



Example 2: Simple Line Chart

```
from matplotlib import pyplot as plt

years = [1950, 1960, 1970, 1980, 1990, 2000, 2010]
gdp = [300.2, 543.3, 1075.9, 2862.5, 5979.6, 10289.7, 14958.3]

# create a line chart, years on x-axis, gdp on y-axis
plt.plot(years, gdp, color='green', marker='o',
linestyle='solid')

# add a title
plt.title("Nominal GDP")

# add a label to the y-axis
plt.ylabel("Billions of $")

# save the figure and display it
plt.savefig('gdp.png', dpi=300)
plt.show()
```



Bar Charts

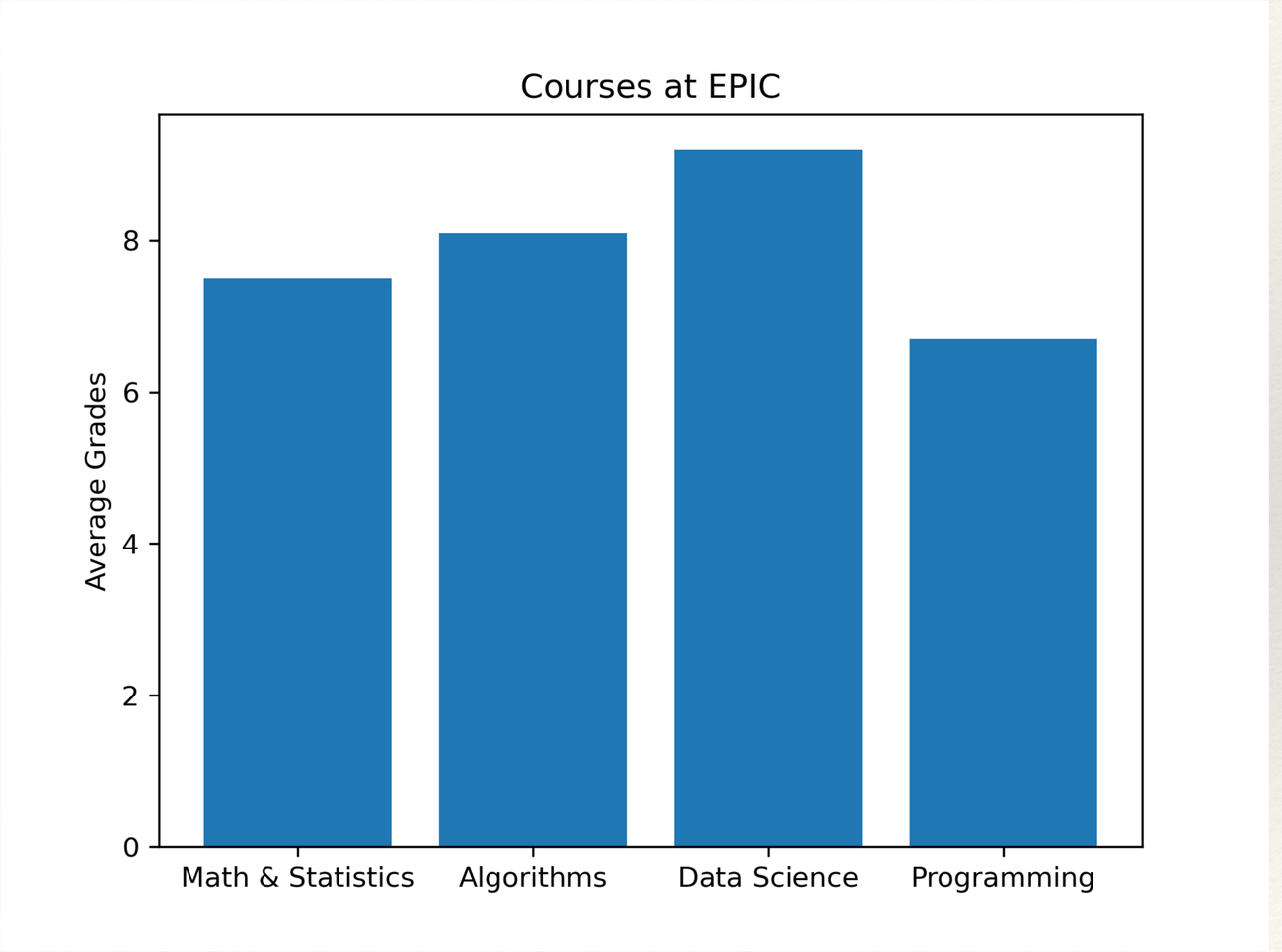
A **bar chart** is a good choice when you want to show how some quantity varies among some discrete set of items.

A **bar chart** can also be a good choice for plotting histograms of bucketed numeric values in order to visually explore how the values are distributed.

Use **plt.bar**

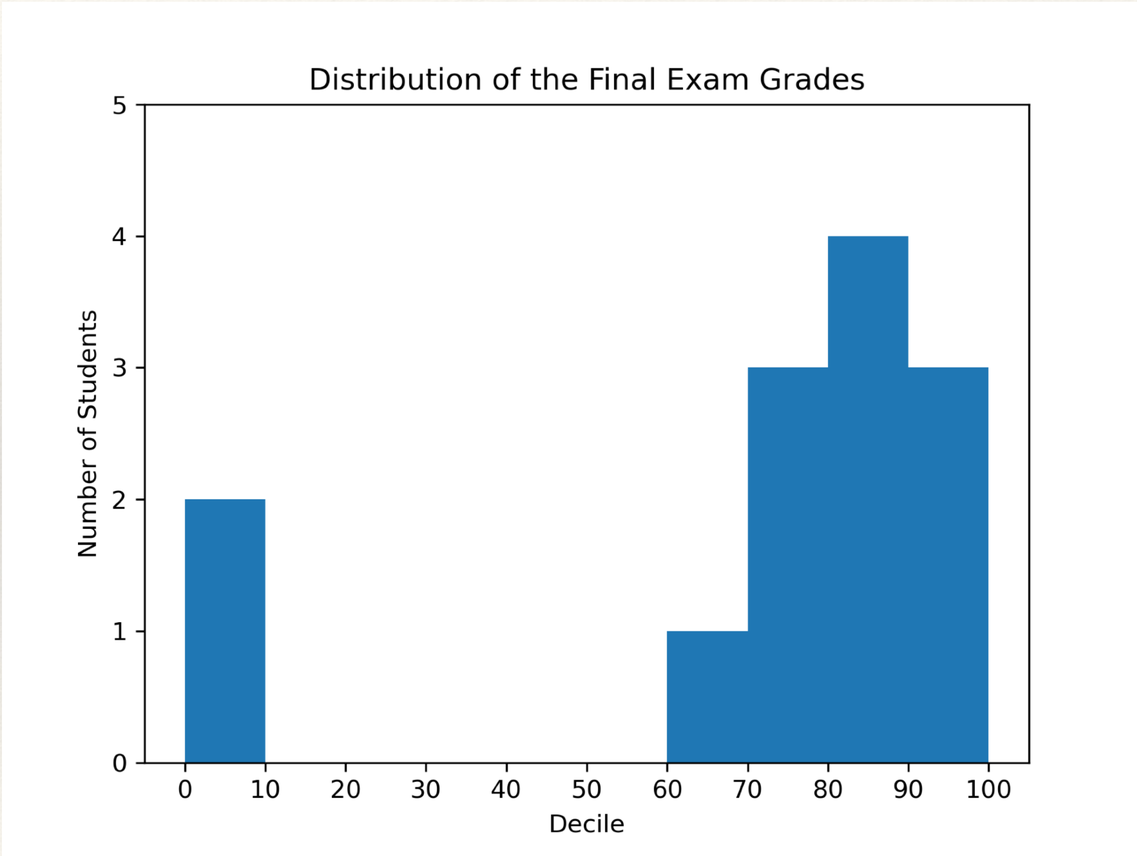
Example 3: Bar Chart

```
courses = ["Math & Statistics", "Algorithms", "Data Science", "Programming"]
avg_grades = [7.5, 8.1, 9.2, 6.7]

# plot bars with left x-coordinates [0, 1, 2, 3], heights [avg_grades]
plt.bar(range(len(courses)), avg_grades)

plt.title("Courses at EPIC") # add a title
plt.ylabel("Average Grades") # label the y-axis

# label x-axis with course names at bar centres
plt.xticks(range(len(courses)), courses)

# save the figure and display it
plt.savefig('avggrades', dpi=300)
plt.show()
```



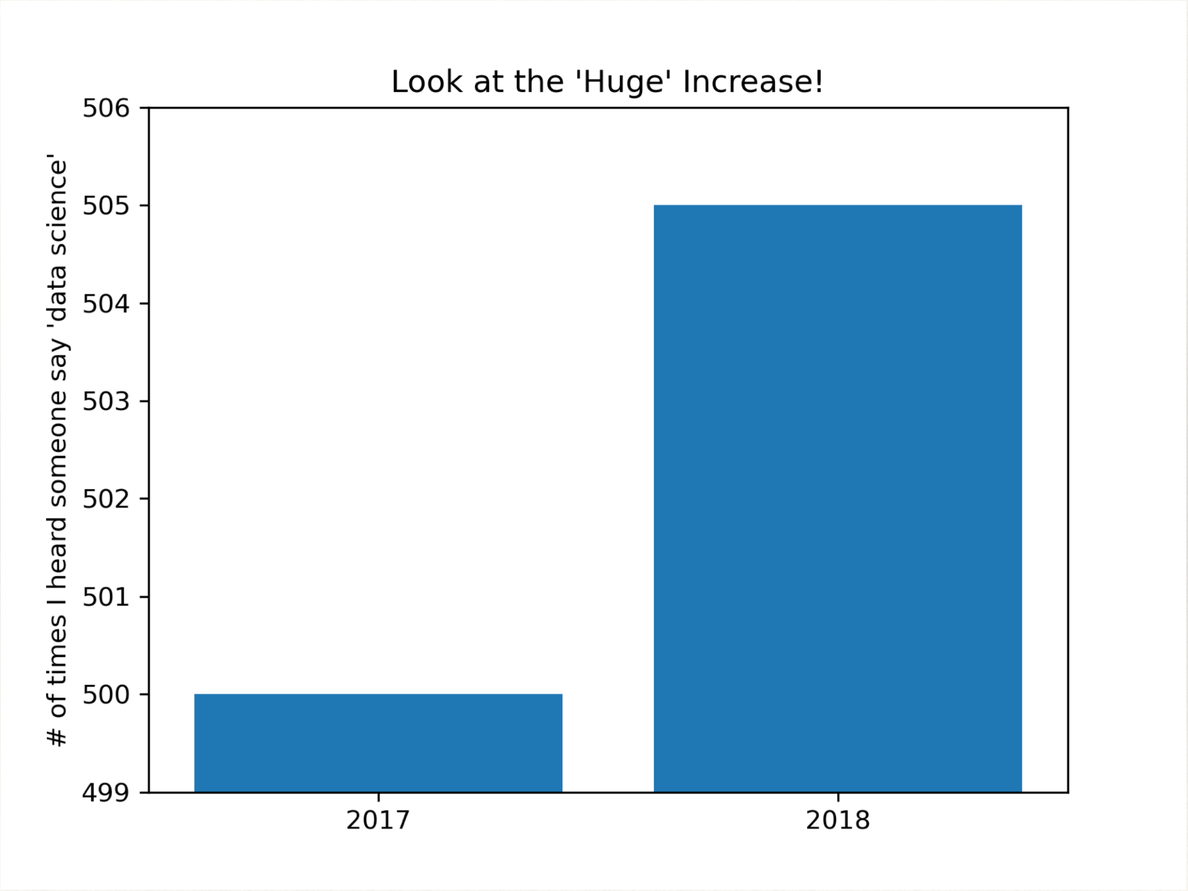
Example 4: Bar Chart

```
from collections import Counter
grades = [83,95,91,87,70,0,85,82,100,67,73,77,0]

# Bucket grades by decile, but put 100 in with the 90s
histogram = Counter(min(grade // 10 * 10, 90) for grade in grades)

plt.bar([x + 5 for x in histogram.keys()], # shift each bar to the right by 5
        histogram.values(),    # give each bar its correct height
        10)                    # give each bar a width of 8
plt.axis([-5, 105, 0, 5])      # x-axis from -5 to 105, y-axis from 0 to 5

plt.xticks([10 * i for i in range(11)])    # x-axis labels at 0, 10, ..., 100
plt.xlabel("Decile")
plt.ylabel("Number of Students")
plt.title("Distribution of the Final Exam Grades")
```



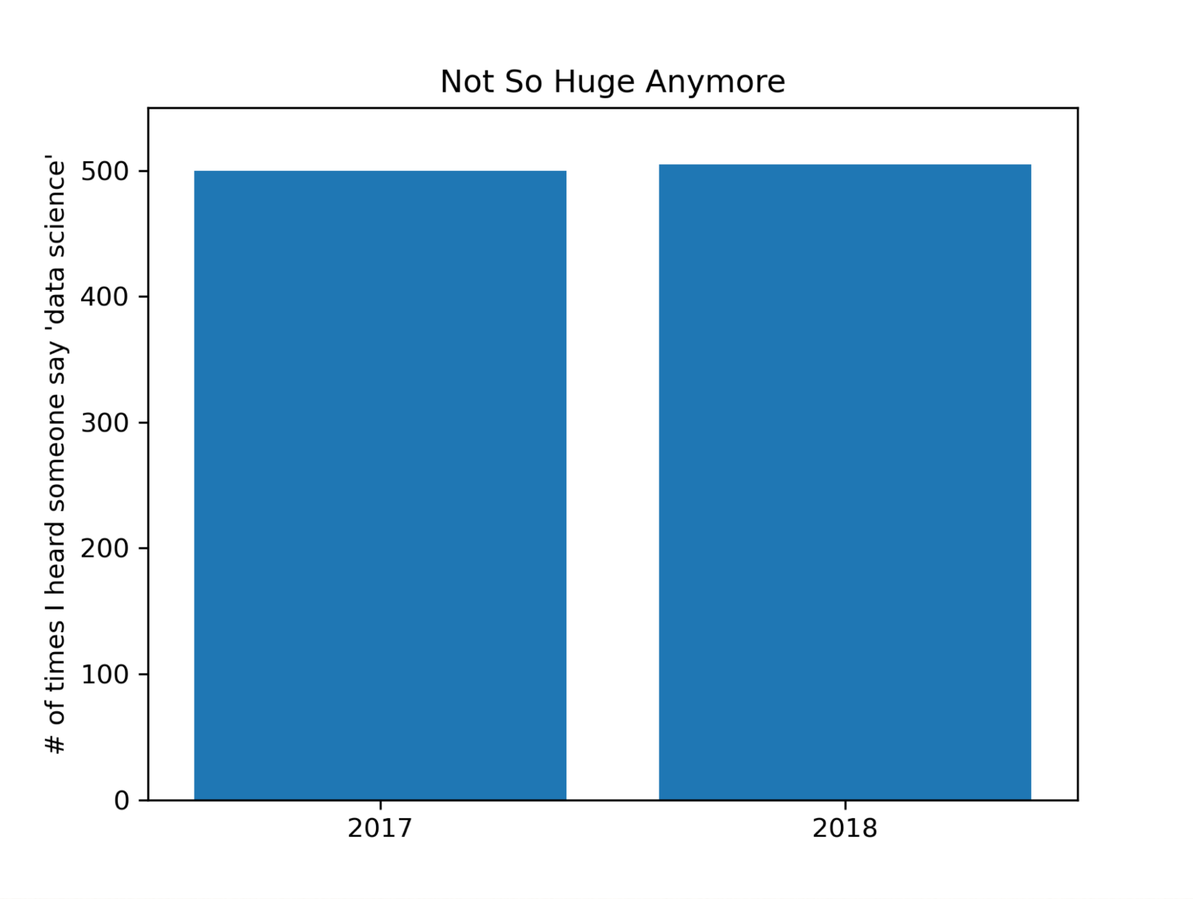
Example 5: Bar Chart (Bad)

```
# When creating bar charts it is considered especially bad form for your y-axis not to start at 0, since this is an easy way to mislead people

mentions = [500, 505]
years = [2017, 2018]
plt.bar(years, mentions, 0.8)
plt.xticks(years)
plt.ylabel("# of times I heard someone say 'data science'")

# if you don't do this, matplotlib will label the x-axis 0, 1
# and then add a +2.013e3 off in the corner (bad matplotlib!)
plt.ticklabel_format(useOffset=False)

# misleading y-axis only shows the part above 500
plt.axis([2016.5, 2018.5, 499, 506])
plt.title("Look at the 'Huge' Increase!")
```



Example 5: Bar Chart (Good)

```
# When creating bar charts it is considered especially bad form for your y-axis not to start at 0, since this is an easy way to mislead people

years = [2017, 2018]
mentions = [500, 505]

plt.bar(years, mentions, 0.8)
plt.xticks(years)
plt.ylabel("# of times I heard someone say 'data science'")

# if you don't do this, matplotlib will label the x-axis 0, 1
# and then add a +2.013e3 off in the corner (bad matplotlib!)
plt.ticklabel_format(useOffset=False)

# misleading y-axis only shows the part above 500
plt.axis([2016.5, 2018.5, 0, 550])
plt.title("Not So Huge Anymore")
```



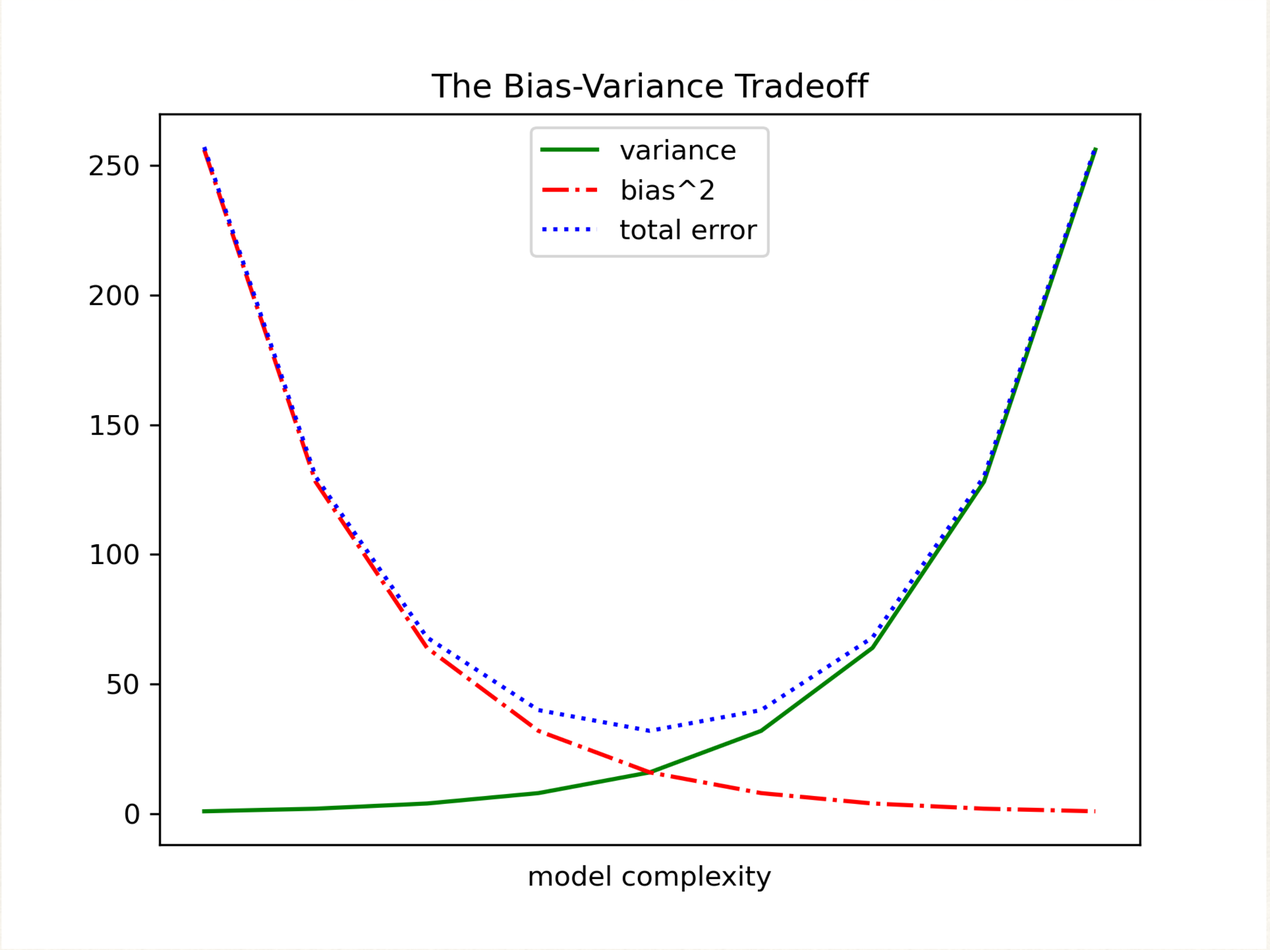
Line Charts

A **line chart** is a good choice for showing trends.

Use **plt.plot**

Example 6: Line Chart

```
variance = [1, 2, 4, 8, 16, 32, 64, 128, 256]
bias_squared = [256, 128, 64, 32, 16, 8, 4, 2, 1]
total_error = [x + y for x, y in zip(variance, bias_squared)]
xs = [i for i, _ in enumerate(variance)]

# We can make multiple calls to plt.plot
# to show multiple series on the same chart
plt.plot(xs, variance, 'g-', label='variance') # green solid line
plt.plot(xs, bias_squared, 'r-.', label='bias^2') # red dot-dashed line
plt.plot(xs, total_error, 'b:', label='total error') # blue dotted line

# Because we've assigned labels to each series,
# we can get a legend for free (loc=9 means "top center")
plt.legend(loc=9)
plt.xlabel("model complexity")
plt.xticks([])
plt.title("The Bias-Variance Tradeoff")
```



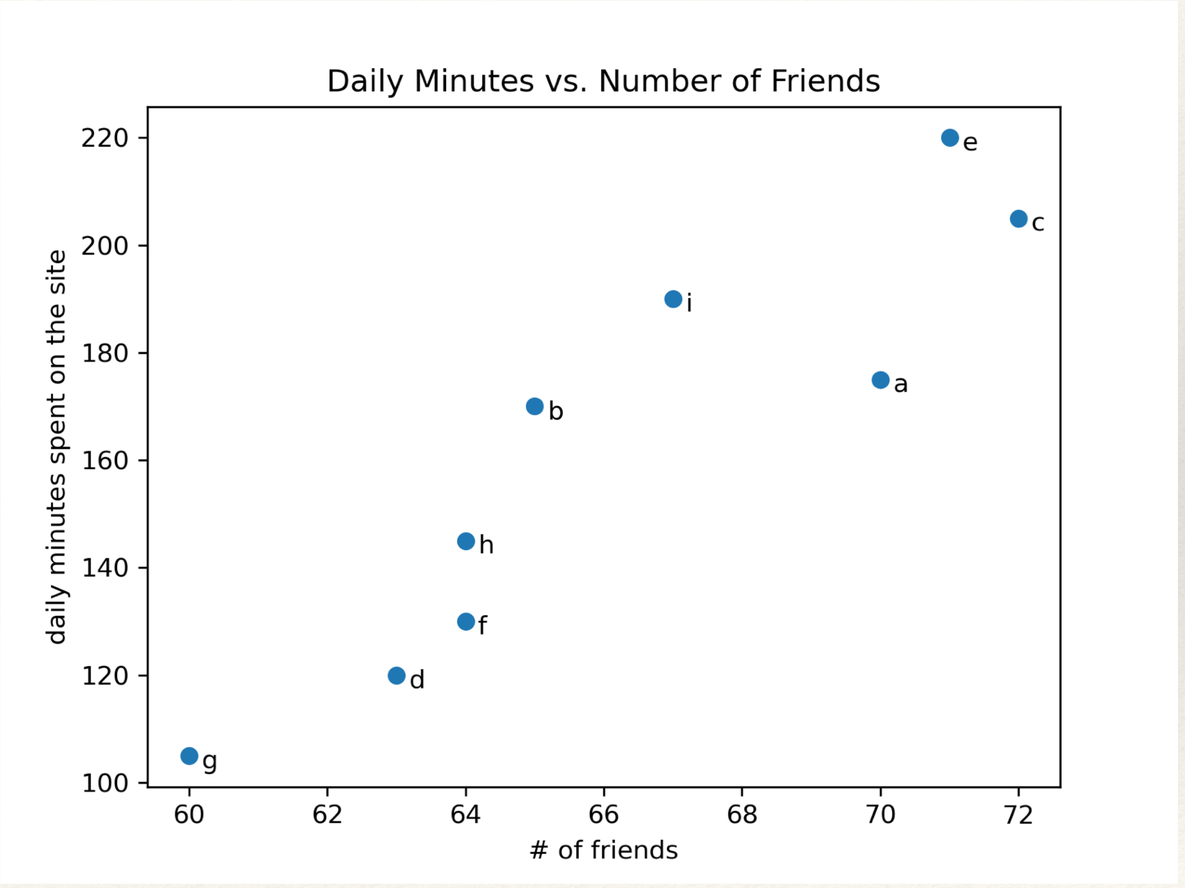
Scatterplots

A **scatterplot** is the right choice for visualizing the relationship between two paired sets of data.

Use **plt.scatter**

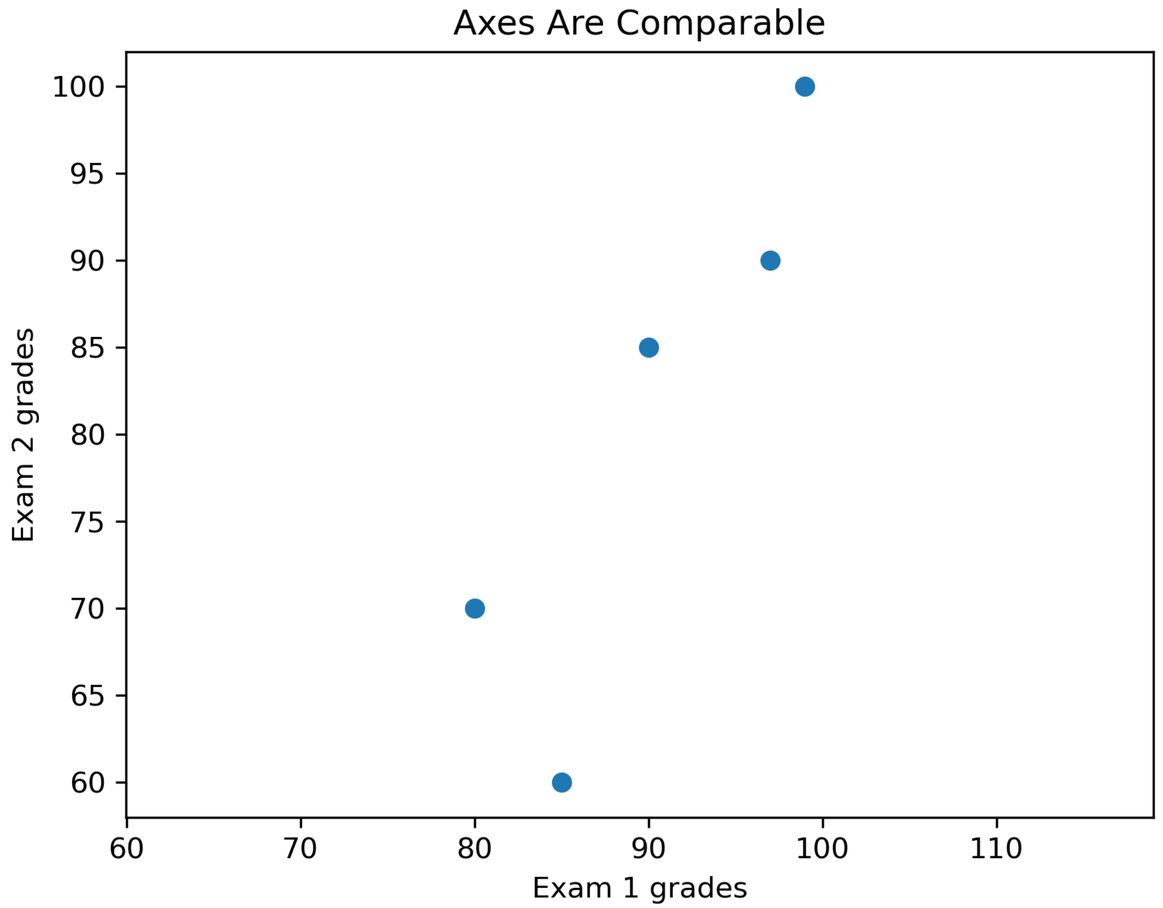
Example 7: Scatterplots

```
# Relationship between the number of friends your users have
# and the number of minutes they spend on the site every day

friends = [ 70,  65,  72,  63,  71,  64,  60,  64,  67]
minutes = [175, 170, 205, 120, 220, 130, 105, 145, 190]
labels =  ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i']
plt.scatter(friends, minutes)

# label each point
for label, friend_count, minute_count in zip(labels, friends, minutes):
    plt.annotate(label,
        xy=(friend_count, minute_count), # Put the label with its point
        xytext=(5, -5),                  # but slightly offset
        textcoords='offset points')

plt.title("Daily Minutes vs. Number of Friends")
plt.xlabel("# of friends")
plt.ylabel("daily minutes spent on the site")
```



Example 8: Scatterplots

```
exam_1_grades = [ 99, 90, 85, 97, 80]
exam_2_grades = [100, 85, 60, 90, 70]
plt.scatter(exam_1_grades, exam_2_grades)

# If we include a call to plt.axis("equal"), the plot more
# accurately shows that most of the variation occurs on exam 2
plt.axis("equal")
plt.title("Axes Are Comparable")
plt.xlabel("Exam 1 grades")
plt.ylabel("Exam 2 grades")
```



Figures and Subplots

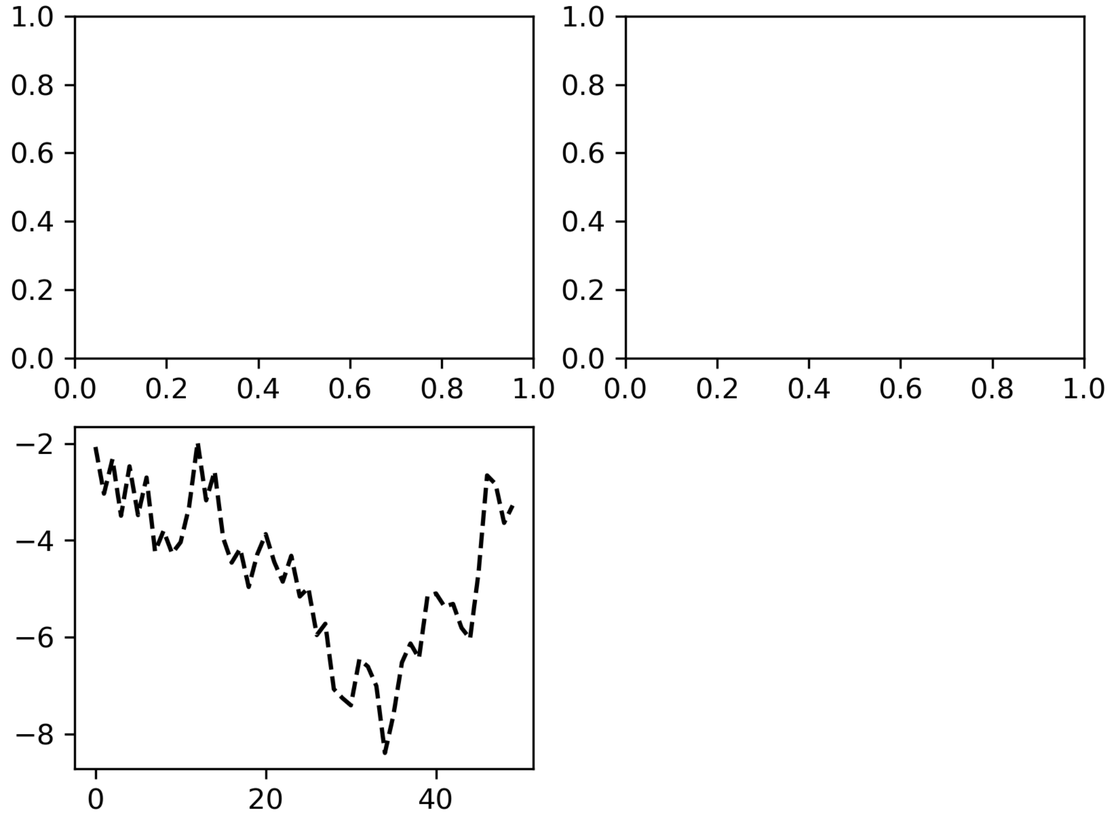
Plots in matplotlib reside within a **Figure** object.

You can create a new figure with **plt.figure**

You can create one or more subplots
using **add_subplot**

Example 9: Figures and Subplots

```
fig = plt.figure()

# Make a figure to be 2 × 2 (so up to four plots in total),
# number subplots (1, 2, 3)
ax1 = fig.add_subplot(2, 2, 1)
ax2 = fig.add_subplot(2, 2, 2)
ax3 = fig.add_subplot(2, 2, 3)

# If you issue a plotting command like plt.plot, then matplotlib
# draws on the last figure
# and subplot used (creating one if necessary)
import numpy as np
plt.plot(np.random.randn(50).cumsum(), 'k--')
```

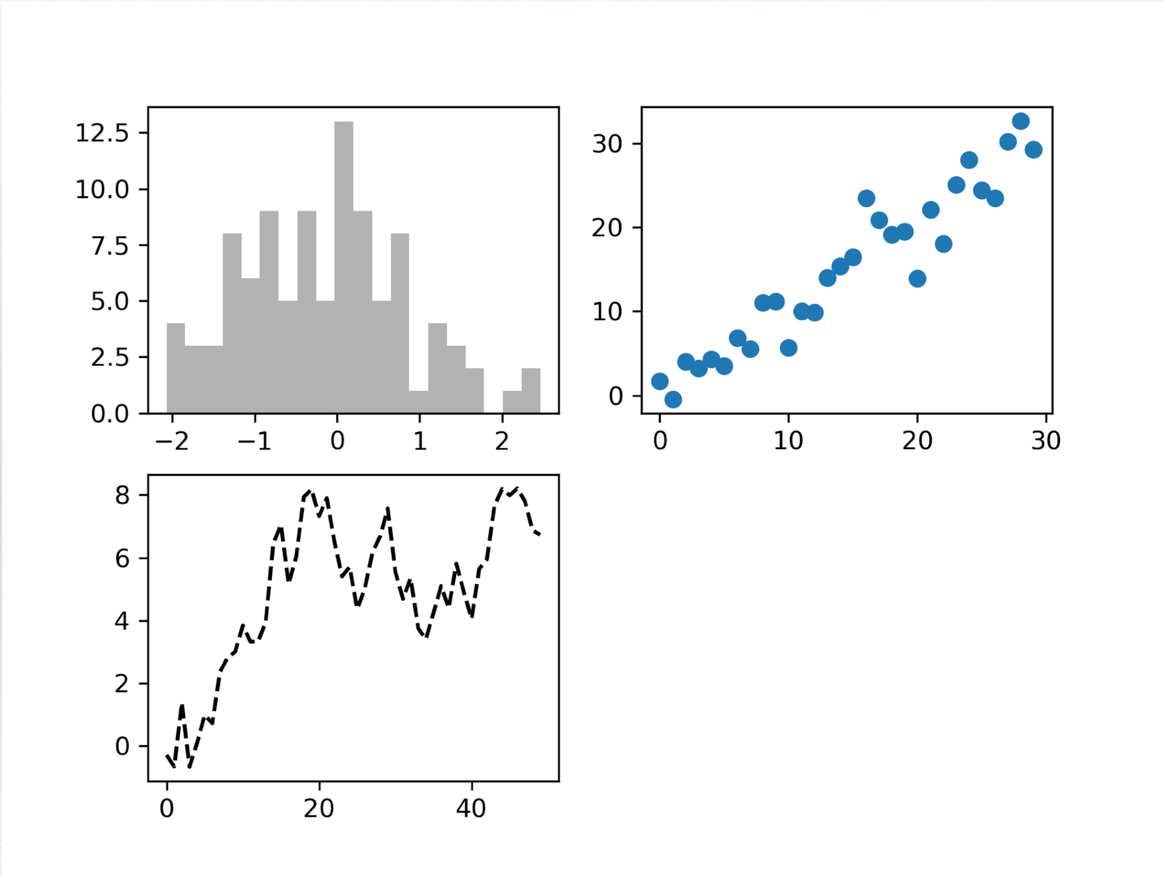


Example 10: Figures and Subplots

The objects returned by fig.add_subplot here are AxesSubplot objects, on which you can directly plot on the other empty subplots by calling each one's instance method:

_ = ax1.hist(np.random.randn(100), bins=20, color='k', alpha=0.3)

ax2.scatter(np.arange(30), np.arange(30) + 3 * np.random.randn(30))

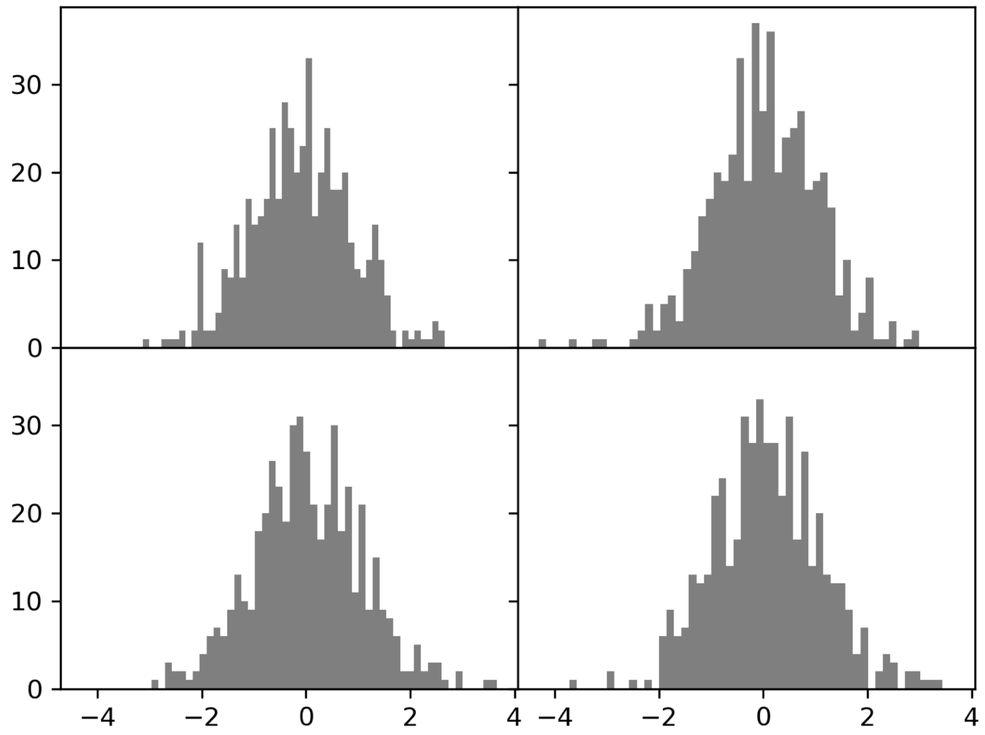


pyplot.subplots options

Argument	Description
`nrows`	Number of rows of subplots
`ncols`	Number of columns of subplots
`sharex`	All subplots should use the same x-axis ticks (adjusting the `xlim` will affect all subplots)
`sharey`	All subplots should use the same y-axis ticks (adjusting the `ylim` will affect all subplots)
`subplot_kw`	Dict of keywords passed to `add_subplot` call used to create each subplot
`**fig_kw`	Additional keywords to `subplots` are used when creating the figure, such as `plt.subplots(2, 2, figsize=(8, 6))`

Example 11: Adjusting the spacing around subplots

```
# subplots_adjust(left=None, bottom=None, right=None, top=None,
                  wspace=None, hspace=None)

fig, axes = plt.subplots(2, 2, sharex=True, sharey=True)
for i in range(2):
    for j in range(2):
        axes[i, j].hist(np.random.randn(500), bins=50, color='k', alpha=0.5)

plt.subplots_adjust(wspace=0, hspace=0)
```

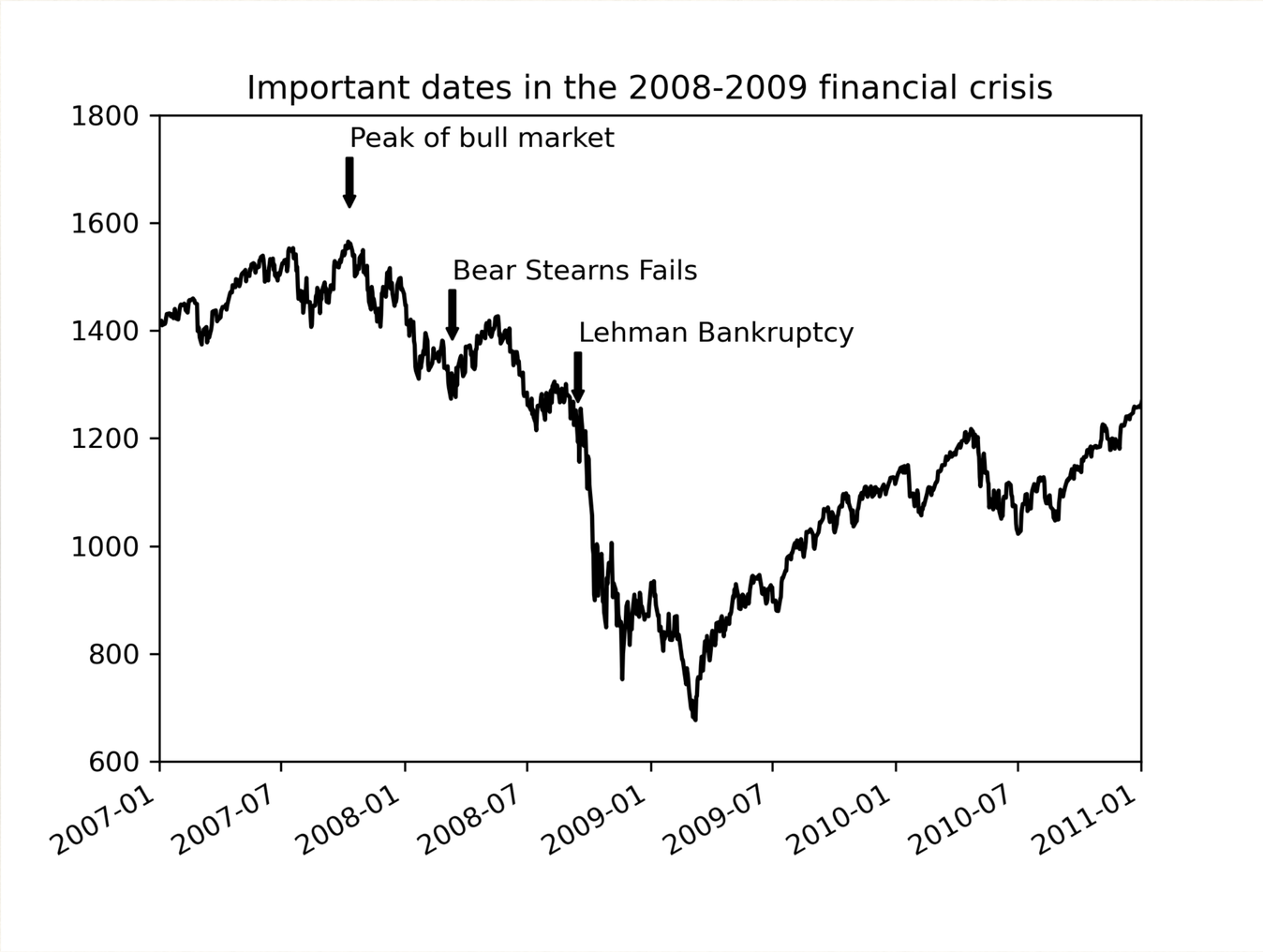


Annotations and Drawing on a Subplot

In addition to the standard plot types, you may wish to draw your own plot annotations, which could consist of text, arrows, or other shapes. You can add annotations and text using the **text**, **arrow**, and **annotate** functions.

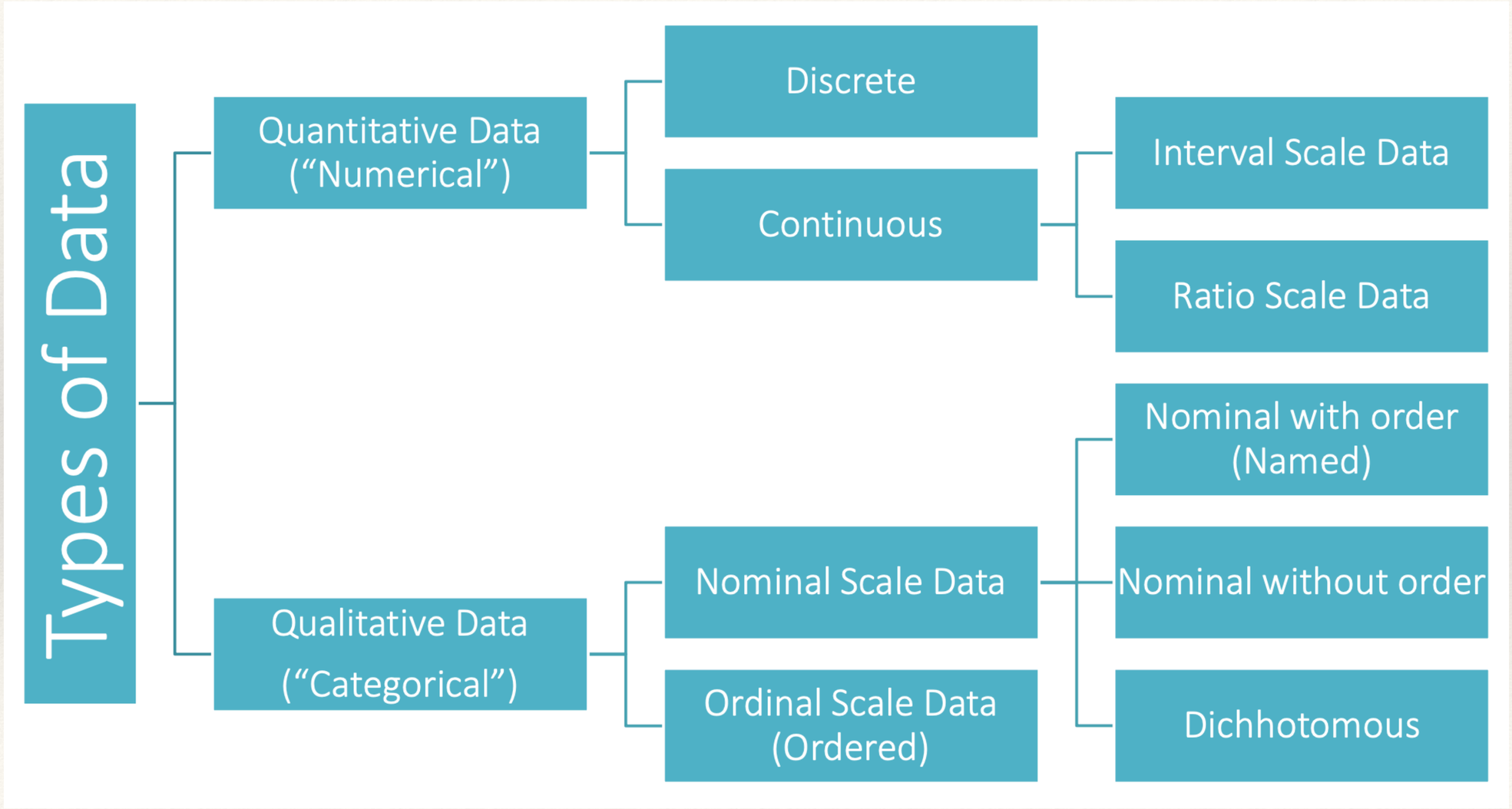
Example 12: Annotations and Drawing on a Subplot

```
import pandas as pd
from datetime import datetime
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)

data = pd.read_csv('spx.csv', index_col=0, parse_dates=True)
spx = data['SPX']
spx.plot(ax=ax, style='k-')
crisis_data = [
    (datetime(2007, 10, 11), 'Peak of bull market'),
    (datetime(2008, 3, 12), 'Bear Stearns Fails'),
    (datetime(2008, 9, 15), 'Lehman Bankruptcy')
]
for date, label in crisis_data:
    ax.annotate(label, xy=(date, spx.asof(date) + 75),
                xytext=(date, spx.asof(date) + 225),
                arrowprops=dict(facecolor='black', headwidth=4, width=2,
                                headlength=4),
                horizontalalignment='left', verticalalignment='top')
# Zoom in on 2007-2010
ax.set_xlim(['1/1/2007', '1/1/2011'])
ax.set_ylim([600, 1800])
ax.set_title('Important dates in the 2008-2009 financial crisis')
```

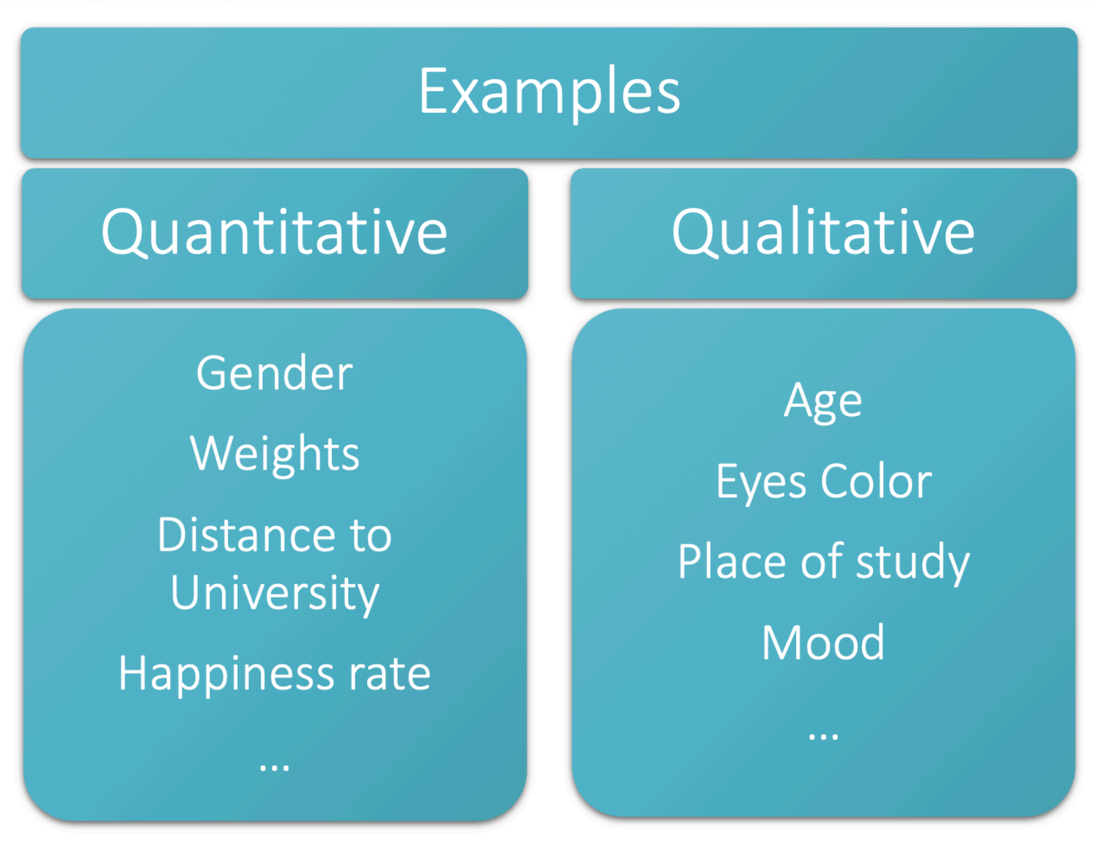


Data Types in Statistics

Major Data Types in Statistics



Quantitative vs. Qualitative

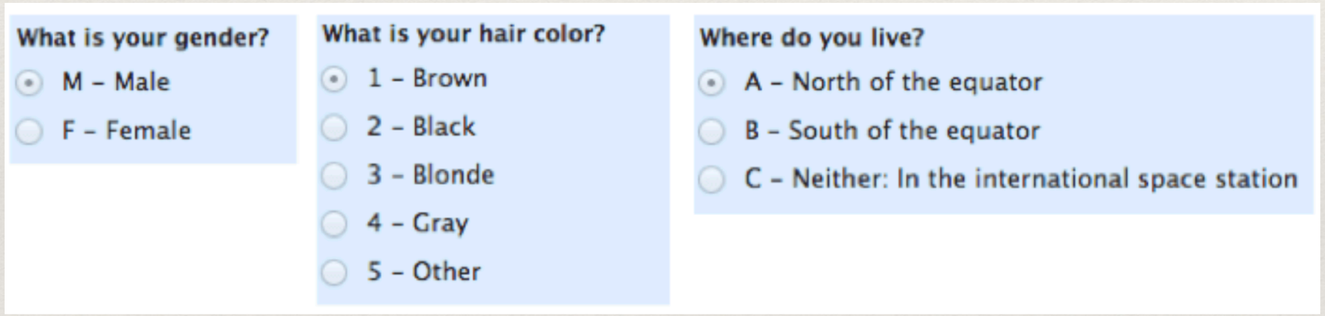


- Quantitative: numerical data
- Qualitative: no numbers

Quantitative vs. Qualitative

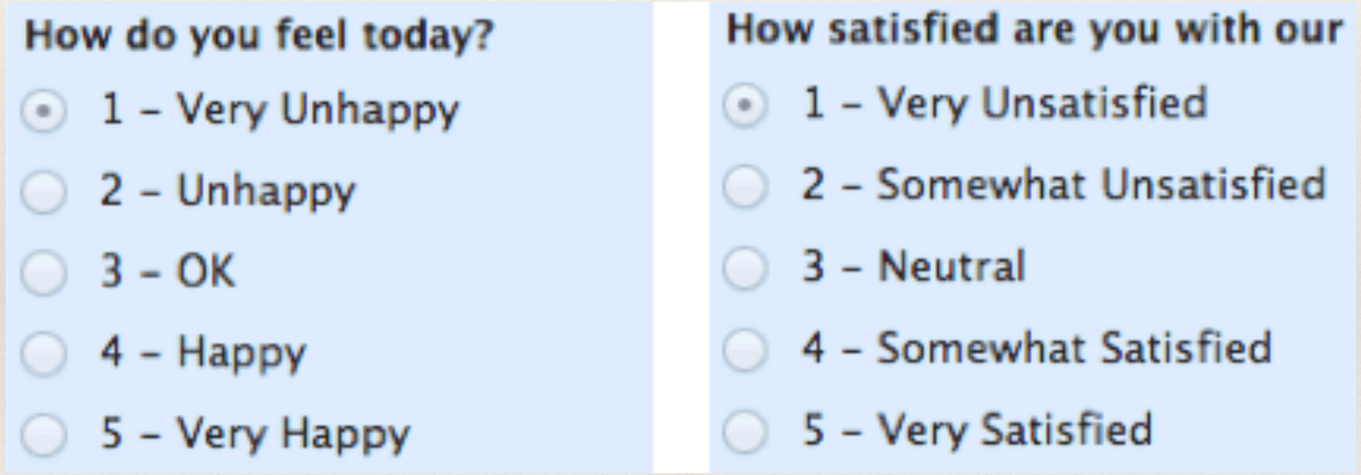
Data Type	Details	Strengths	Weaknesses	Gathered in
Qualitative	Written data e.g. interview notes, open answers on a survey	Lots of detail Can develop new insights	Difficult to analyse Subjectivity (can use triangulation to improve) Time consuming	Interviews Questionnaires Observations Content Analysis Grounded Theory
Quantitative	Numerical Data e.g. height, IQ score, scores on a test, Analysed using statistics	Easy to analyse Objective Reliable Can present in graph form	Lack of detail Reductionist Hard to get insights	Questionnaires Experiments Observations

Qualitative Data: Nominal

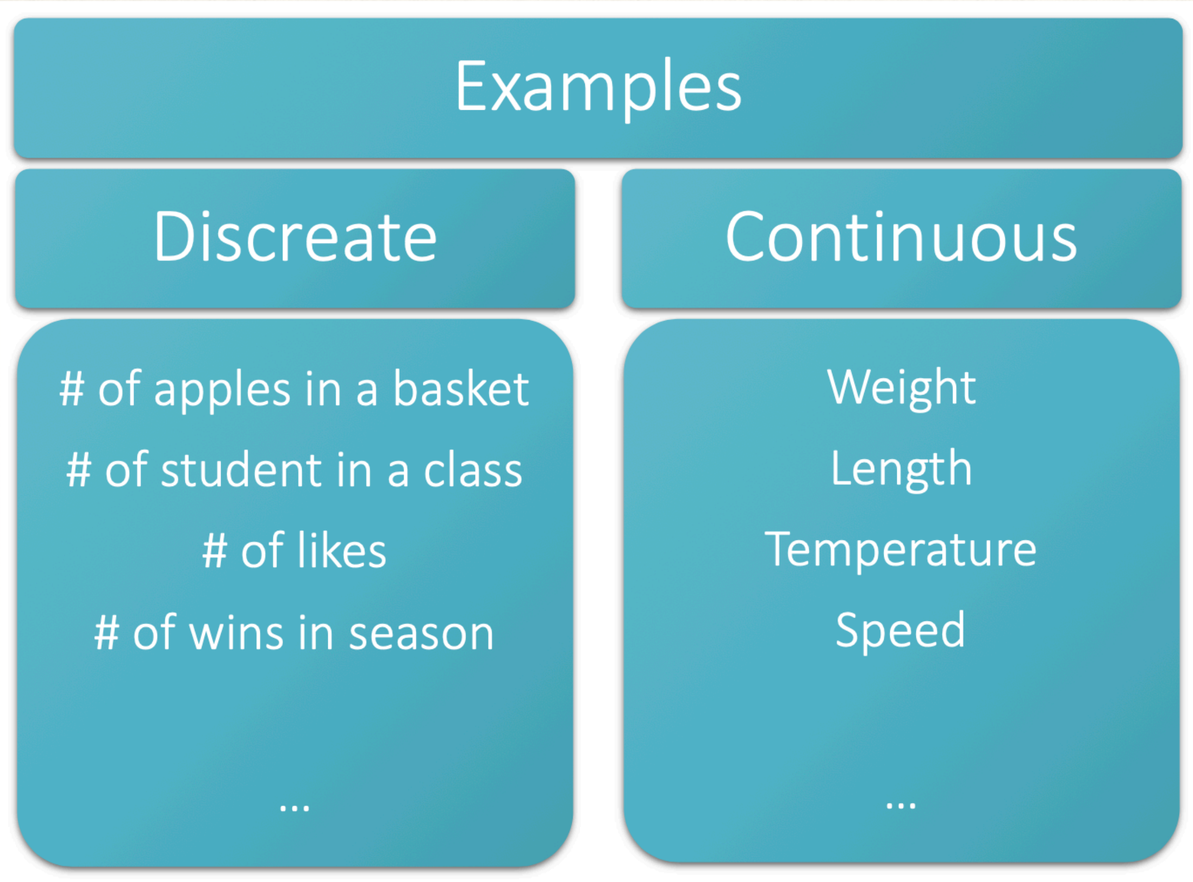


- Dichotomous
- Nominal with order / nominal without order
- Categorized / non categorized

Qualitative Data: Ordinal



Quantitative Data: Discrete vs. Continuous



- **Discrete**: numbers that cannot be divided
- **Continuous**: numbers, that can be broken into finer and finer units (usually within range)

Quantitative Data: Interval vs. Ratio

- **Interval scales** hold no true zero and can represent values below zero. For example, you can measure temperatures below 0 degrees Celsius, such as -10 degrees.
- **Ratio variables**, on the other hand, never fall below zero. Height and weight measure from 0 and above, but never fall below it.
- In an **interval scale**, the data collected can be added, subtracted, and multiplied. The scale allows computing the degree of difference but not the ratio between them.
- A **ratio scale** permits not only addition, subtraction, and multiplication but also division. That is, you can calculate the ratio of the values.

Basic operations and functions

Provides:	Nominal	Ordinal	Interval	Ratio
The "order" of values is known		yes	yes	yes
"Counts", "Freq. or Distr."	yes	yes	yes	yes
Mode	yes	yes	yes	yes
Median		yes	yes	yes
Mean			yes	yes
Can quantify the diff. between each value			yes	yes
Can add or substract values			yes	yes
Can multiply and divide values				yes
Has "true zero"				yes

- **Nominal** variables are used to "name," or label a series of values.
- **Ordinal** scales provide good information about the order of choices, such as in a customer satisfaction survey.
- **Interval** scales give us the order of values + the ability to quantify the difference between each one.
- **Ratio** scales give us the ultimate–order, interval values, plus the ability to calculate ratios since a "true zero" can be defined.

Basic operations and functions

Date Type	Basic operations	Applied statistics	Data analysis methods
Nominal	Equality	The frequency of each class, modal class	chi-sq. criteria for hypothesis testing about diastribution
Ordinal	Determination of differences in properties in relation to "more", "less"	Ordinal statistics: median, quantiles	Methods based on the use of ranks (rank correlation, non-parametric hypotheses testing)
Interval	Determining whether a value belongs to a given interval	Mean, standart error, mixed moments, correlation	All methods of data analysis
Ratio	Determining relationship equality	All previous, also harmonic mean, coeff. of variation	All methods of data analysis

Thank you!