

CSC380: Principles of Data Science

Data Analysis, Collection, and Visualization 1

Xinchen Yu

On D2L - Content:

- HW03 solutions posted
- Midterm practical problems posted
- HW04 posted

Pandas

Open source library for data handling and manipulation in high-performance environments.



Installation If you are using Anaconda package manager,

```
conda install pandas
```

Or if you are using PyPi (pip) package manager,

```
pip install pandas
```

See Pandas documentation for more detailed instructions
https://pandas.pydata.org/docs/getting_started/install.html

Primary data structure : Essentially a table

The diagram illustrates a DataFrame as a table. It features a table with 7 rows and 5 columns. The columns are labeled 'Name', 'Team', 'Number', 'Position', and 'Age'. The rows are indexed from 0 to 6. Annotations include: 'Columns' with arrows pointing to the column headers; 'Rows' with arrows pointing to the row indices; and 'Data' with a bracket pointing to the data cells. Some cells are highlighted with purple boxes: 'Jonas Jerebko', '8.0', 'NaN', 'PG', and 'NaN'.

	<i>Name</i>	<i>Team</i>	<i>Number</i>	<i>Position</i>	<i>Age</i>
0	Avery Bradley	Boston Celtics	0.0	PG	25.0
1	John Holland	Boston Celtics	30.0	SG	27.0
2	Jonas Jerebko	Boston Celtics	8.0	PF	29.0
3	Jordan Mickey	Boston Celtics	NaN	PF	21.0
4	Terry Rozier	Boston Celtics	12.0	PG	22.0
5	Jared Sullinger	Boston Celtics	7.0	C	NaN
6	Evan Turner	Boston Celtics	11.0	SG	27.0

Q: how is it different from numpy array?

- Numpy arrays are more efficient
- Pandas dataframes are more flexible

Create and print an entire DataFrame

```
# import pandas as pd
import pandas as pd

# list of strings
lst = ['Geeks', 'For', 'Geeks', 'is',
       'portal', 'for', 'Geeks']

# Calling DataFrame constructor on list
df = pd.DataFrame(lst)
print(df)
```

0	
0	Geeks
1	For
2	Geeks
3	is
4	portal
5	for
6	Geeks

Can create named columns using dictionary

```
import pandas as pd

# initialise data of lists.
data = {'Name': ['Tom', 'nick', 'krish', 'jack'],
        'Age': [20, 21, 19, 18]}

# Create DataFrame
df = pd.DataFrame(data)

# Print the output.
print(df)
```

	Name	Age
0	Tom	20
1	nick	21
2	krish	19
3	jack	18

all data must have the same length



Select columns to print by name,

```
# Import pandas package
import pandas as pd

# Define a dictionary containing employee data
data = {'Name': ['Jai', 'Princi', 'Gaurav', 'Anuj'],
        'Age': [27, 24, 22, 32],
        'Address': ['Delhi', 'Kanpur', 'Allahabad', 'Kannauj'],
        'Qualification': ['Msc', 'MA', 'MCA', 'Phd']}

# Convert the dictionary into DataFrame
df = pd.DataFrame(data)

# select two columns
print(df[['Name', 'Qualification']])
```

	Name	Qualification
0	Jai	Msc
1	Princi	MA
2	Gaurav	MCA
3	Anuj	Phd

access columns by name, not the column index!

DataFrame : Selecting Columns

9

```
[35]: import pandas as pd
data = {'Name': ['tom', 'nick'], 'Age': [10,20]}
df = pd.DataFrame(data)
```

```
[36]: df[['Name']]
```

```
[36]:
```

	Name
0	tom
1	nick

```
[37]: df['Name']
```

```
[37]: 0    tom
      1    nick
      Name: Name, dtype: object
```

```
[38]: type(df[['Name']]), type(df['Name'])
```

```
[38]: (pandas.core.frame.DataFrame, pandas.core.series.Series)
```

pandas.Series

`class pandas.Series(data=None, index=None, dtype=None, name=None, copy=False, fastpath=False)` [\[source\]](#)

One-dimensional ndarray with axis labels (including time series).

Labels need not be unique but must be a hashable type. The object supports both integer- and label-based indexing and provides a host of methods for performing operations involving the index. Statistical methods from ndarray have been overridden to automatically exclude missing data (currently represented as NaN).

still a DataFrame

essentially, a 'named' array

```
import pandas as pd
import numpy as np

# Define a dictionary containing employee data
data = {'Name': ['Jai', 'Princi', 'Gaurav', 'Anuj'],
        'Age': [27, 24, 22, 32],
        'Address': ['Delhi', 'Kanpur', 'Allahabad', 'Kannauj'],
        'Qualification': ['Msc', 'MA', 'MCA', 'Phd']}

# Convert the dictionary into DataFrame
df = pd.DataFrame(data)

# Print rows 1 & 2
row = df.loc[1:2]
print(row)
```

2nd and 3rd row!

1:2 includes 2! annoying! this is not python standard!!!

Output

	Name	Age	Address	Qualification
1	Princi	24	Kanpur	MA
2	Gaurav	22	Allahabad	MCA

(still a DataFrame)

```
[6]: import pandas as pd
      data = {'Name': ['tom', 'nick'], 'Age': [10,20]}
      df = pd.DataFrame(data)
```

```
[19]: df.loc[1:1]
```

```
[19]:
```

	Name	Age
1	nick	20

```
[20]: df.loc[1]
```

```
[20]: Name    nick
      Age      20
      Name: 1, dtype: object
```

```
[21]: type(df.loc[1:1]), type(df.loc[1])
```

```
[21]: (pandas.core.frame.DataFrame, pandas.core.series.Series)
```

- `df.loc[1:1]` is DataFrame object
- `df.loc[1]` is a series

`head()` and `tail()` select rows from beginning / end

```
import pandas as pd
import numpy as np

# Define a dictionary containing employee data
data = {'Name': ['Jai', 'Princi', 'Gaurav', 'Anuj'],
        'Age': [27, 24, 22, 32],
        'Address': ['Delhi', 'Kanpur', 'Allahabad', 'Kannauj'],
        'Qualification': ['Msc', 'MA', 'MCA', 'Phd']}

# Convert the dictionary into DataFrame
df = pd.DataFrame(data)

# Print first / last rows
first2 = df.head(2)
last2 = df.tail(2)
print(first2)
print('\n', last2)
```

Output

	Name	Age	Address	Qualification
0	Jai	27	Delhi	Msc
1	Princi	24	Kanpur	MA

	Name	Age	Address	Qualification
2	Gaurav	22	Allahabad	MCA
3	Anuj	32	Kannauj	Phd

Easy reading / writing of standard formats,

index ↓

Output

```
df = pd.read_json("data.json")
print(df)
df.to_csv("data.csv", index=False)
df_csv = pd.read_csv("data.csv")
print(df_csv.head(2))
```

	Duration	Pulse	Maxpulse	Calories
0	60	110	130	409.1
1	60	117	145	479.0
2	60	103	135	340.0
3	45	109	175	282.4
4	45	117	148	406.0
..
164	60	105	140	290.8
165	60	110	145	300.4
166	60	115	145	310.2
167	75	120	150	320.4
168	75	125	150	330.4

[169 rows x 4 columns]

	Duration	Pulse	Maxpulse	Calories
0	60	110	130	409.1
1	60	117	145	479.0

example: twitter api returns search results in json format.

Working with DataFrames outside of Pandas can be tricky,

```
df['Duration']
```

Q: does it return a DataFrame object or Series object?

We can easily convert to built-in types,
for example to a list.

```
0      60
1      60
2      60
3      45
4      45
..
164    60
165    60
166    60
167    75
168    75
Name: Duration, Length: 169, dtype: int64
```

```
L = df['Duration'].to_list()
print(L)
```

```
[60, 60, 60, 45, 45, 60, 60, 45, 30, 60, 60, 60, 60, 60, 60, 60, 60, 60, 45, 60, 45, 60, 45, 60, 45, 60, 60, 60, 60, 60,
60, 60, 45, 60, 60, 60, 60, 60, 60, 60, 45, 45, 60, 60, 60, 80, 60, 60, 30, 60, 60, 45, 2
0, 45, 210, 160, 160, 45, 20, 180, 150, 150, 20, 300, 150, 60, 90, 150, 45, 90, 45, 45, 120, 270, 30, 45, 30, 120, 4
5, 30, 45, 120, 45, 20, 180, 45, 30, 15, 20, 20, 30, 25, 30, 90, 20, 90, 90, 90, 30, 30, 180, 30, 90, 210, 60, 45, 1
5, 45, 60, 60, 60, 60, 60, 60, 30, 45, 60, 60, 60, 60, 60, 60, 90, 60, 60, 60, 60, 60, 20, 45, 45, 45, 20, 60, 6
0, 45, 45, 60, 45, 60, 60, 30, 60, 60, 60, 60, 30, 60, 60, 60, 60, 60, 30, 30, 45, 45, 45, 60, 60, 60, 75, 75]
```

Or, to a numpy array.

```
[6]: import pandas as pd  
data = {'Name': ['tom', 'nick'], 'Age': [10, 20]}  
df = pd.DataFrame(data)
```

```
[29]: df
```

```
[29]:
```

	Name	Age
0	tom	10
1	nick	20

```
[31]: df.to_numpy()
```

```
[31]: array([[ 'tom', 10],  
          [ 'nick', 20]], dtype=object)
```

```
[40]: df['Name'].to_numpy()
```

```
[40]: array(['tom', 'nick'], dtype=object)
```

to_numpy(): defined on Index, Series, and DataFrame objects

Easily compute summary statistics on data

```
print('Min: ', df['Duration'].min())  
print('Max: ', df['Duration'].max())  
print('Median: ', df['Duration'].median())
```

```
Min: 15  
Max: 300  
Median: 60.0
```

Can also count occurrences of
unique values,

```
df['Duration'].value_counts()
```



```
60    79  
45    35  
30    16  
20     9  
90     8  
150    4  
120    3  
180    3  
15     2  
75     2  
160    2  
210    2  
270    1  
25     1  
300    1  
80     1  
Name: Duration, dtype: int64
```

```
s = df['Duration'].value_counts()  
s[60]=79.
```



```
[42]: import pandas as pd
      data = {'Name': ['tom', 'nick'], 'Age': [10, 20], 'Height': [6.2, 5.5]}
      df = pd.DataFrame(data)
      df
```


```
[42]:
```

	Name	Age	Height
0	tom	10	6.2
1	nick	20	5.5

```
[43]: df.describe()
```

```
[43]:
```

	Age	Height
count	2.000000	2.000000
mean	15.000000	5.850000
std	7.071068	0.494975
min	10.000000	5.500000
25%	12.500000	5.675000
50%	15.000000	5.850000
75%	17.500000	6.025000
max	20.000000	6.200000



use describe() to get a summary of the data

Many database operations are available

- You can specify index, which can speed up some operations
- You can do 'join'
- You can do 'where' clause to filter the data
- You can do 'group by'



🔍 Search the docs ...

Installation

Package overview

Getting started tutorials



What kind of data does pandas handle?

How do I read and write tabular data?

How do I select a subset of a **DataFrame** ?

How to create plots in pandas?

How to create new columns derived from existing columns?

How to calculate summary statistics?

How to reshape the layout of tables?

How to combine data from multiple tables?

How to handle time series data with ease?

How to manipulate textual data?

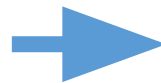
Doing it by yourself helps a lot!

Data Visualization

- Data Visualization
- Data Summarization
- Data Collection and Sampling

```
141 137 134 134 132 130 129 129 131 135 130 128 129 126 128 128 130
138 136 134 134 135 133 131 129 132 139 133 128 130 128 127 129 131
135 135 134 133 133 132 130 128 132 136 134 130 131 132 132 133
133 134 133 132 131 130 130 131 131 129 134 134 130 134 137 134 134
134 134 134 134 133 132 134 138 136 127 135 137 132 136 140 135 139
137 135 136 138 137 135 137 143 142 132 136 138 135 137 138 138 142
139 135 135 138 138 134 135 141 143 133 133 134 135 135 133 138 140
136 137 137 138 141 143 142 144 140 143 142 137 137 139 137 135 136
137 138 136 136 138 140 141 143 140 144 143 139 139 140 138 137 139
137 139 137 136 136 136 137 140 143 146 143 140 141 142 142 143 143
137 140 141 139 138 136 135 137 143 144 142 139 142 144 145 147 146
140 144 144 143 141 137 135 137 139 139 139 139 143 145 146 147 147
145 148 147 145 143 140 139 141 136 138 140 142 147 147 146 147 149
146 148 147 144 143 141 140 143 137 139 142 145 146 145 145 148 147
145 147 146 143 142 140 140 143 138 140 143 143 143 141 143 148 142
145 145 144 144 143 141 141 142 142 145 146 145 144 141 143 150 144
144 143 142 143 143 142 142 144 143 144 143 144 148 144 142 147 145
146 145 144 143 143 143 144 146 144 144 141 146 157 154 144 143 148
149 148 145 144 143 143 144 145 144 146 142 149 167 169 155 146 151
150 149 147 145 142 142 143 143 145 147 143 147 166 175 164 151 152
150 150 149 147 145 145 145 145 147 148 143 142 154 165 160 148 150
152 152 152 150 149 150 150 149 151 151 150 147 146 152 153 147 151
152 153 153 152 151 151 151 150 152 152 156 155 148 149 155 153 152
152 152 152 152 152 151 151 151 152 152 152 153 152 151 152 153 154
152 152 152 152 152 151 151 152 152 152 153 152 151 151 152 154
153 153 153 153 153 153 153 154 154 154 153 153 153 152 150 152 154
153 153 153 153 154 154 154 154 154 154 154 153 153 153 152 153 155
153 153 152 153 154 154 154 154 153 154 154 153 153 153 153 154 157
153 152 152 152 154 155 155 155 153 155 155 154 152 152 152 154 159
```

Encoding



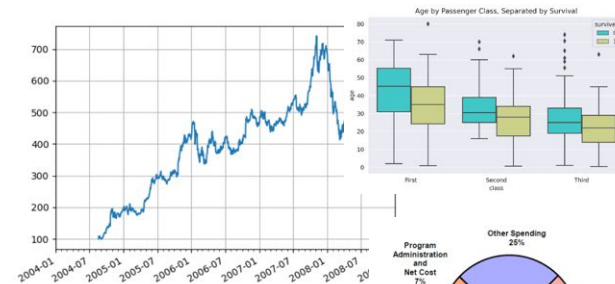
Iterate



Visual
Perception



Understanding

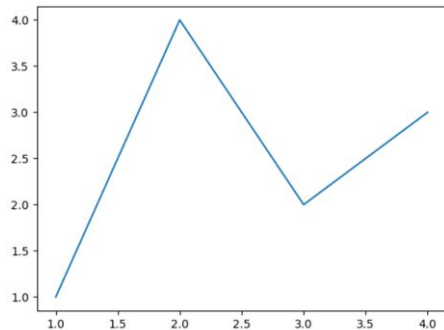


Data visualization in Python...

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

Create a simple figure with an axis object,

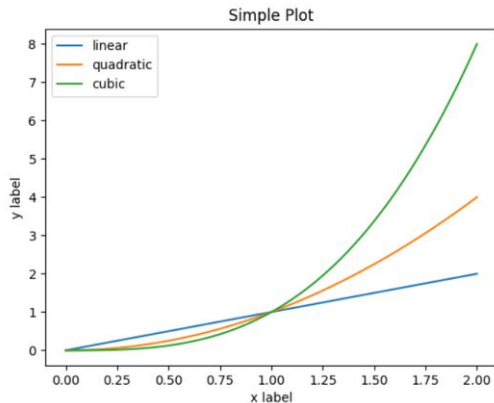
```
fig, ax = plt.subplots() # Create a figure containing a single axes.
ax.plot([1, 2, 3, 4], [1, 4, 2, 3]) # Plot some data on the axes.
```



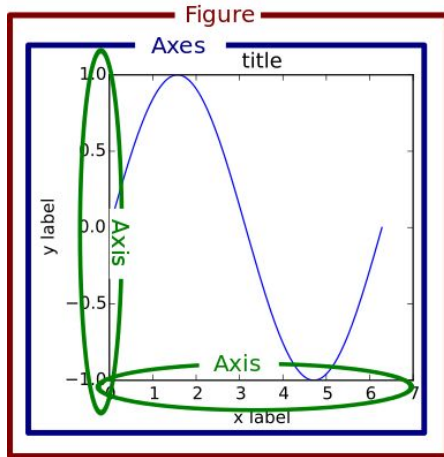
A more complicated plot...

```
x = np.linspace(0, 2, 100)

# Note that even in the OO-style, we use `.pyplot.figure` to create the figure.
fig, ax = plt.subplots() # Create a figure and an axes.
ax.plot(x, x, label='linear') # Plot some data on the axes.
ax.plot(x, x**2, label='quadratic') # Plot more data on the axes...
ax.plot(x, x**3, label='cubic') # ... and some more.
ax.set_xlabel('x label') # Add an x-label to the axes.
ax.set_ylabel('y label') # Add a y-label to the axes.
ax.set_title("Simple Plot") # Add a title to the axes.
ax.legend() # Add a legend.
```



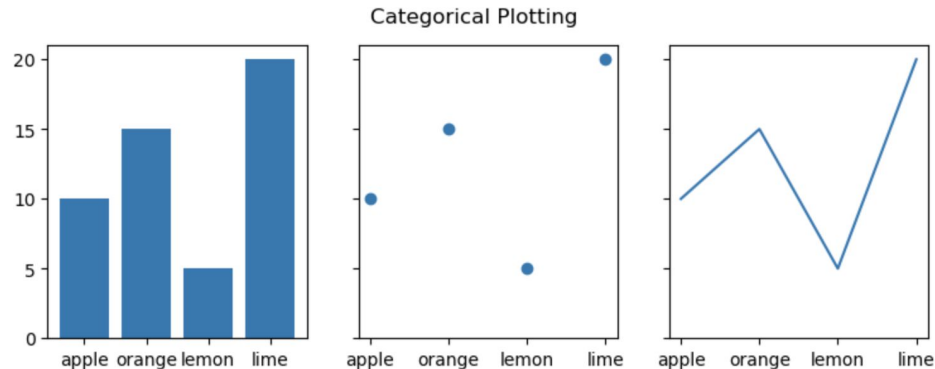
Axes vs Axis

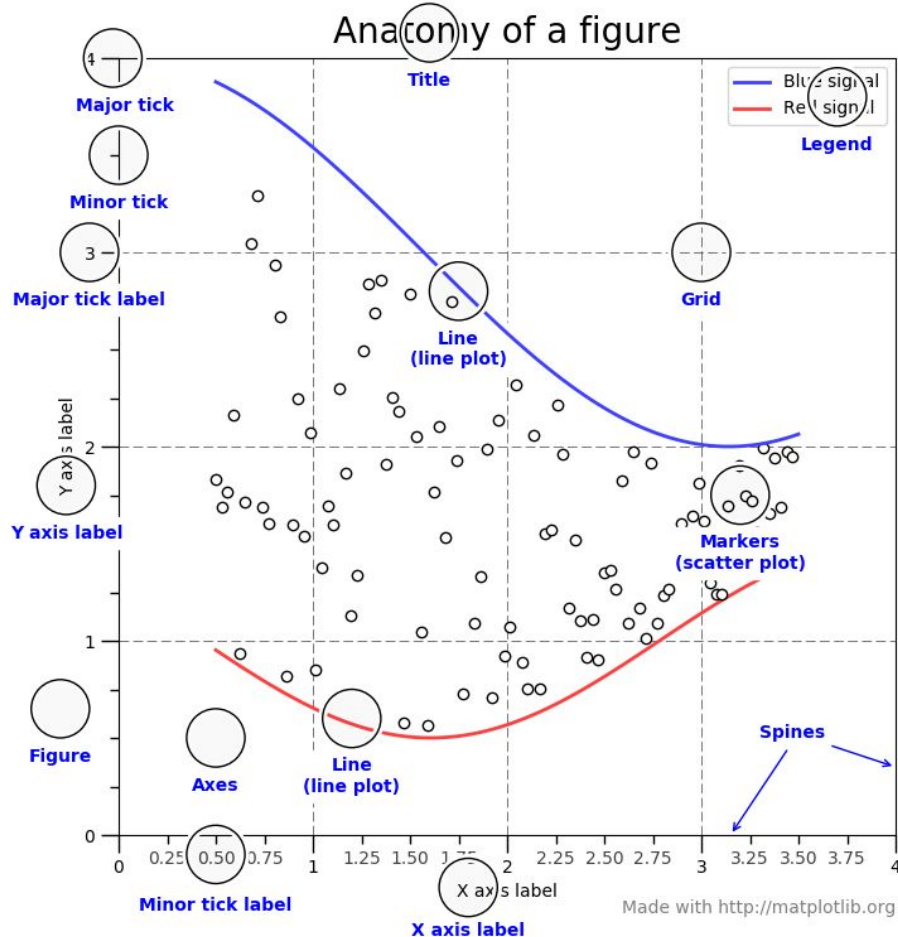


`subplot()` function: draw multiple plots in one figure

```
data = {'apple': 10, 'orange': 15, 'lemon': 5, 'lime': 20}
names = list(data.keys())
values = list(data.values())
```

```
fig, axs = plt.subplots(1, 3, figsize=(9, 3), sharey=True)
axs[0].bar(names, values)
axs[1].scatter(names, values)
axs[2].plot(names, values)
fig.suptitle('Categorical Plotting')
```

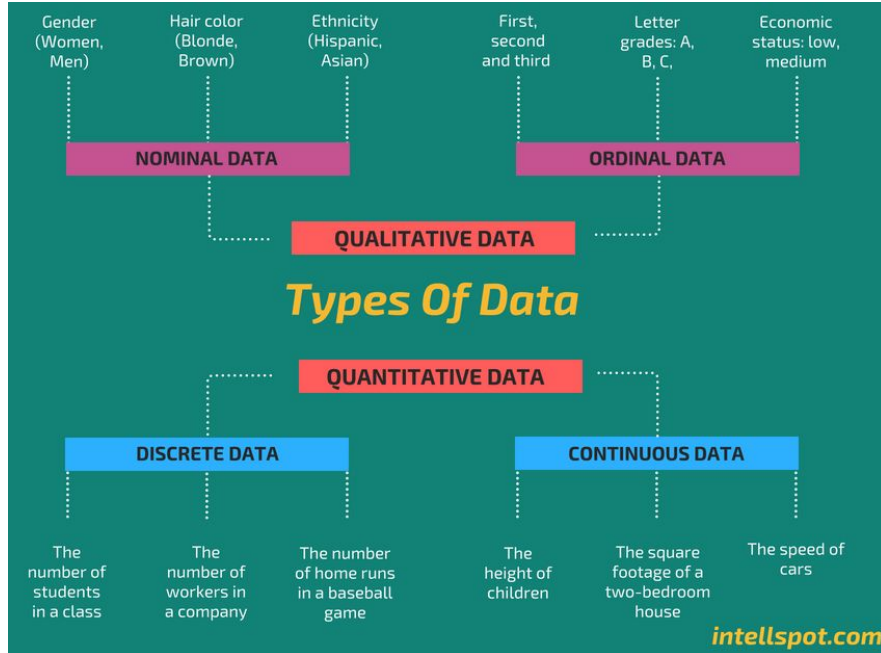




components of a Matplotlib figure

Documentation + tutorials:
<https://matplotlib.org/>

Data come in many forms, each requiring different approaches & models



Qualitative or categorical : can partition data into classes

Quantitative : can perform mathematical operations (e.g., addition, subtraction, ordering)

*We often refer to different types of data as **variables***

Examples

- Roll of a die: 1,2,3,4,5 or 6
- Blood Type: A, B, AB, or O
- Political Party: Democrat, Republican, etc.
- Type of Rock: Igneous, Sedimentary, or Metamorphic
- Word Identity: NP, VP, N, V, Adj, Adv, etc.

Numerical data can be categorical or quantitative depending on context

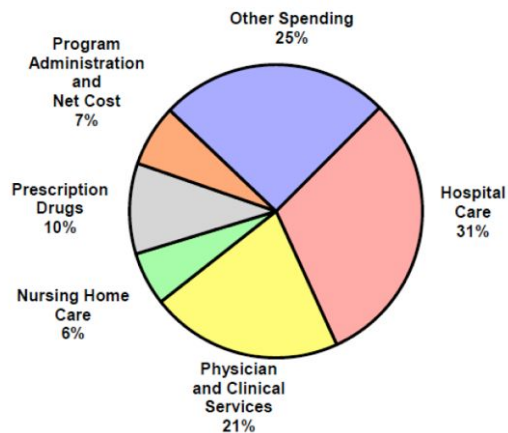


Conversion: Quantitative data can be converted to categorical by defining ranges:

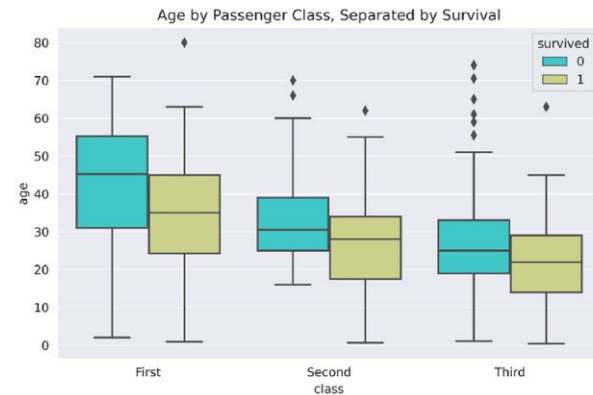
- Small [0, 10cm), Medium [10, 100cm), Large [100cm, 1m), XL [1m, -)
- Low [less than -100dB), Moderate [-100dB, -50dB), Loud [over -50dB)

Visualizing Categorical Variables

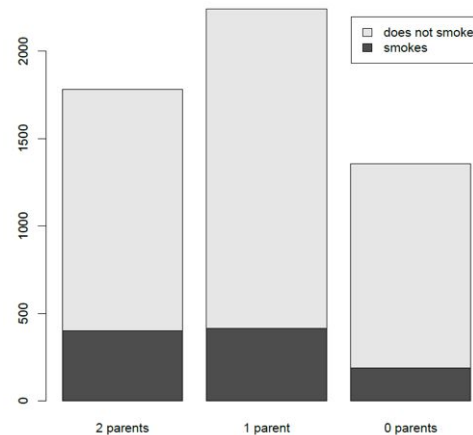
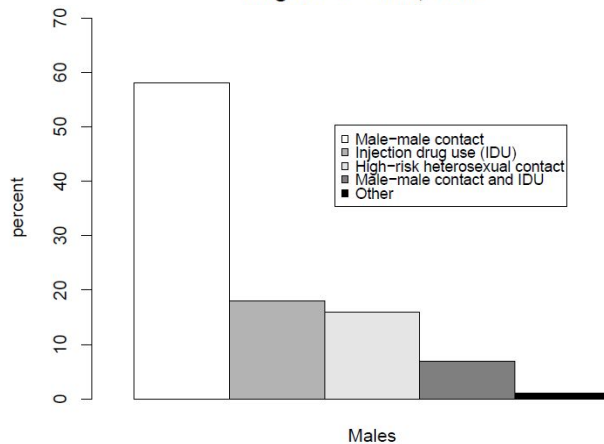
28



	student smokes	student does not smoke	total
2 parents smoke	400	1380	1780
1 parent smokes	416	1823	2239
0 parents smoke	188	1168	1356
total	1004	4371	5375



Proportion of AIDS Cases by Sex and Transmission Category Diagnosed – USA, 2005



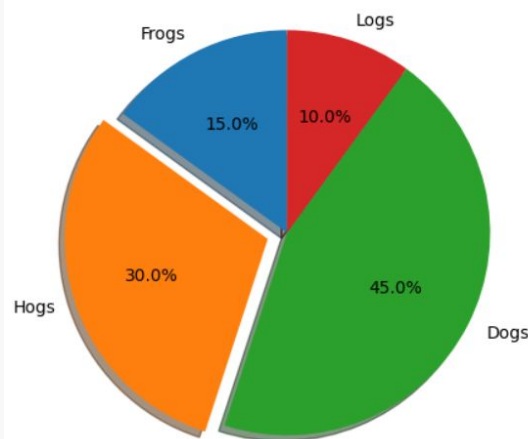
Circular chart divided into sectors, illustrating relative magnitudes in frequencies or percentage. In a pie chart, the area is proportional to the quantity it represents.

```
import matplotlib.pyplot as plt

# Pie chart, where the slices will be ordered and plotted counter-clockwise:
labels = 'Frogs', 'Hogs', 'Dogs', 'Logs'
sizes = [15, 30, 45, 10]
explode = (0, 0.1, 0, 0) # only "explode" the 2nd slice (i.e. 'Hogs')

fig1, ax1 = plt.subplots()
ax1.pie(sizes, explode=explode, labels=labels, autopct='%1.1f%%',
        shadow=True, startangle=90)
ax1.axis('equal') # Equal aspect ratio ensures that pie is drawn as a circle.

plt.show()
```



Maybe the biggest problem with pie charts is that they have been so often done poorly...



bad pie charts



Sign in

All

Images

Videos

News

Shopping

More

Settings

Tools

SafeSearch



wrong



media



example



data visualization

male female

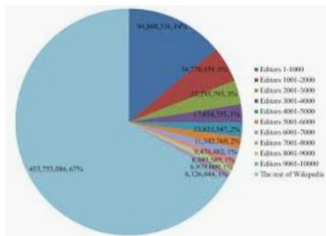
economy florida

2016 presidential election

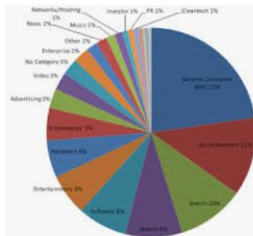
attractive

advanced

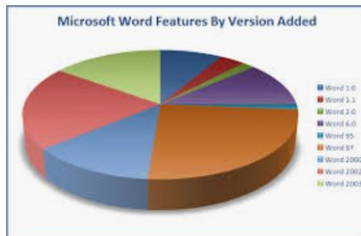
2



Yet another bad pie chart : dataisugly reddit.com



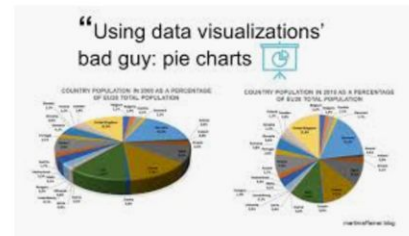
death to pie charts — storytelli... storytellingwithdata.com



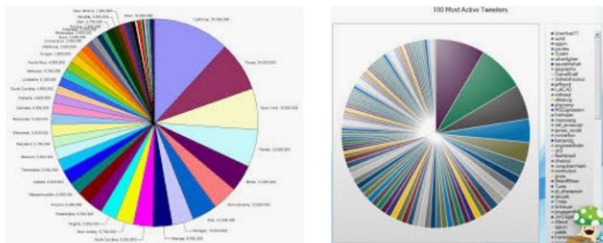
Pie charts: the bad, the worst and the ... visualanalyze.wordpress.com



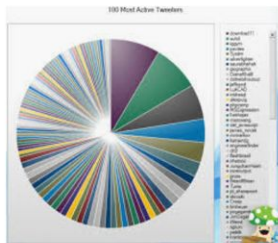
When to use Pie Charts in Dashboards ... excelcampus.com



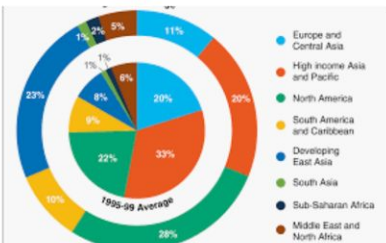
Using data visualizations' bad guy: pie ... martinraffner.blog



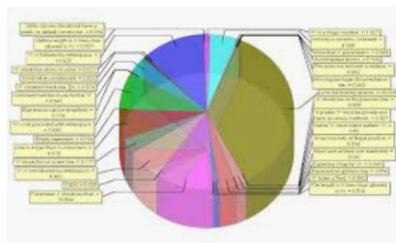
Understanding Pie Charts eagereyes.org



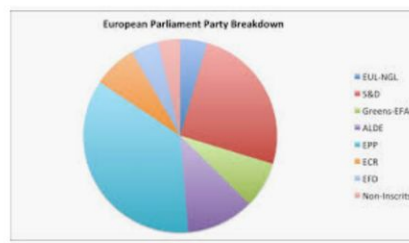
Pie charts: the bad, the worst an... visualanalyze.wordpress.com



Remake: Pie-in-a-Donut Chart - Policy Viz policyviz.com



Pin on Chartjunk Data Visualization pinterest.com



Pie Charts Are The Worst - Business Insider businessinsider.com

We perceive differences in height / length better than area...

`plt.bar()`

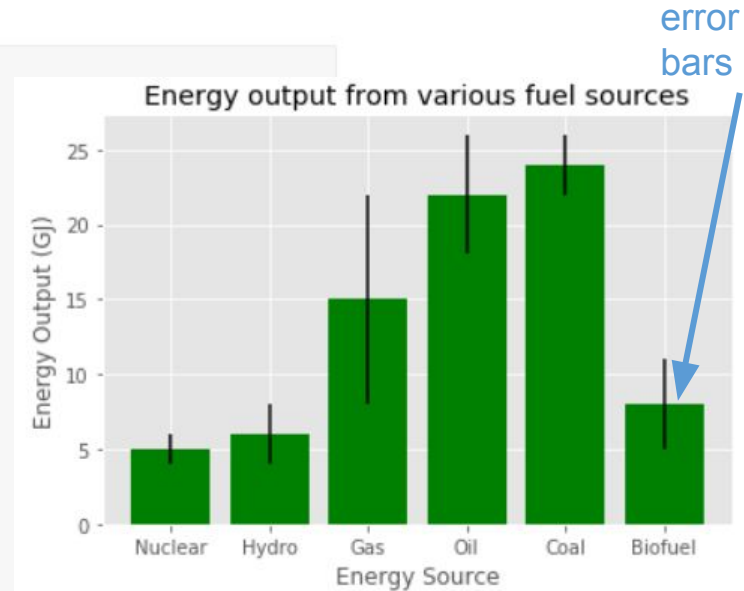
```
x = ['Nuclear', 'Hydro', 'Gas', 'Oil', 'Coal', 'Biofuel']
energy = [5, 6, 15, 22, 24, 8]
variance = [1, 2, 7, 4, 2, 3]

x_pos = [i for i, _ in enumerate(x)]

plt.bar(x_pos, energy, color='green', yerr=variance)
plt.xlabel("Energy Source")
plt.ylabel("Energy Output (GJ)")
plt.title("Energy output from various fuel sources")

plt.xticks(x_pos, x)

plt.show()
```



Horizontal version.

`plt.barh()`

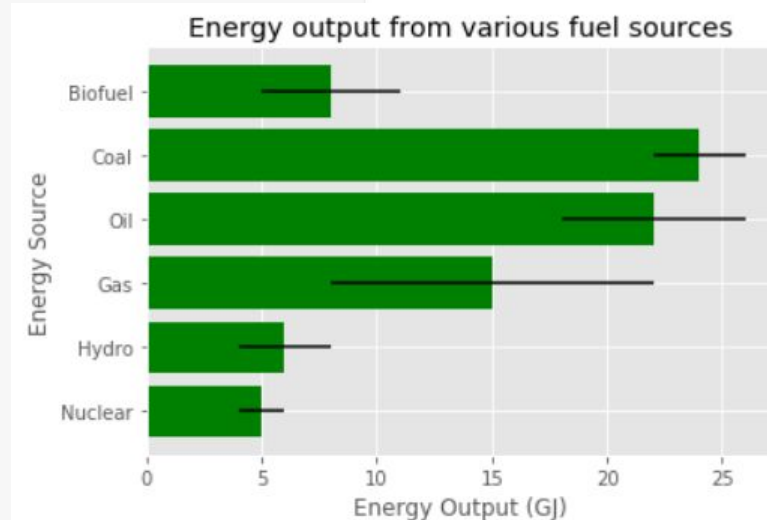
```
x = ['Nuclear', 'Hydro', 'Gas', 'Oil', 'Coal', 'Biofuel']
energy = [5, 6, 15, 22, 24, 8]
variance = [1, 2, 7, 4, 2, 3]

x_pos = [i for i, _ in enumerate(x)]

plt.barh(x_pos, energy, color='green', xerr=variance)
plt.ylabel("Energy Source")
plt.xlabel("Energy Output (GJ)")
plt.title("Energy output from various fuel sources")

plt.yticks(x_pos, x)

plt.show()
```



Multiple groups of bars...

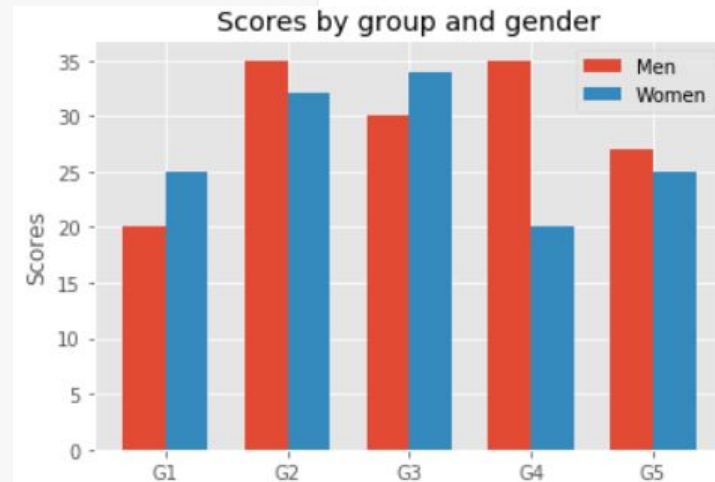
```
import numpy as np

N = 5
men_means = (20, 35, 30, 35, 27)
women_means = (25, 32, 34, 20, 25)

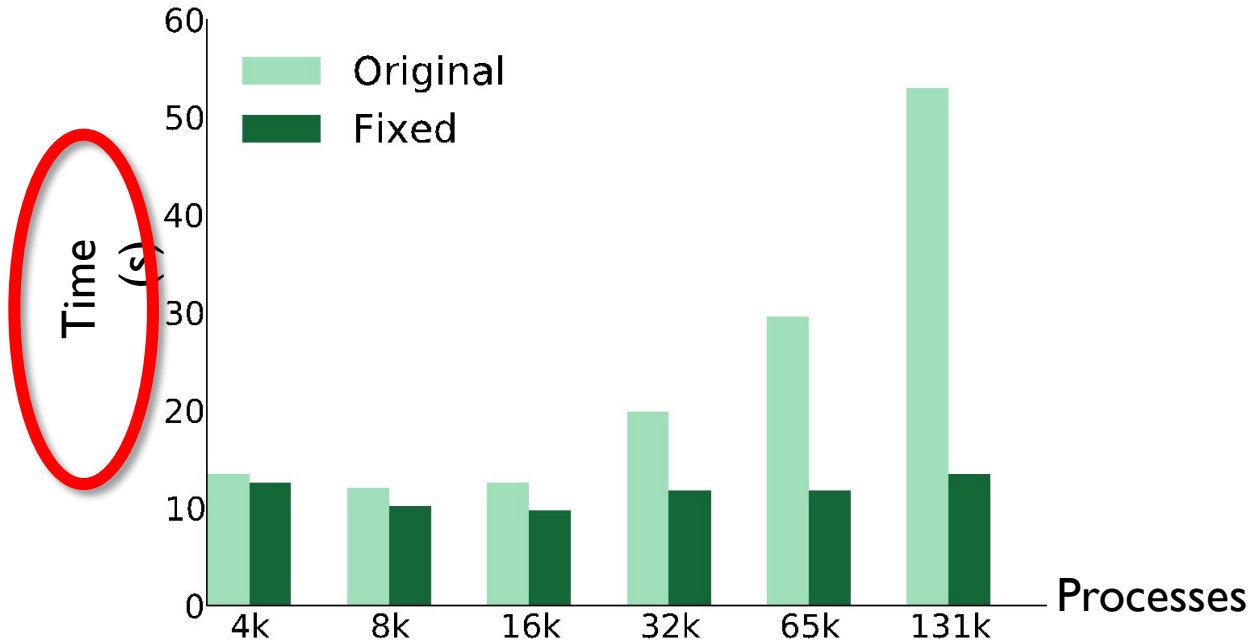
ind = np.arange(N) //
width = 0.35 [1,2,3,4,5]
plt.bar(ind, men_means, width, label='Men')
plt.bar(ind + width, women_means, width,
        label='Women') add the offset
                        here

plt.ylabel('Scores')
plt.title('Scores by group and gender')

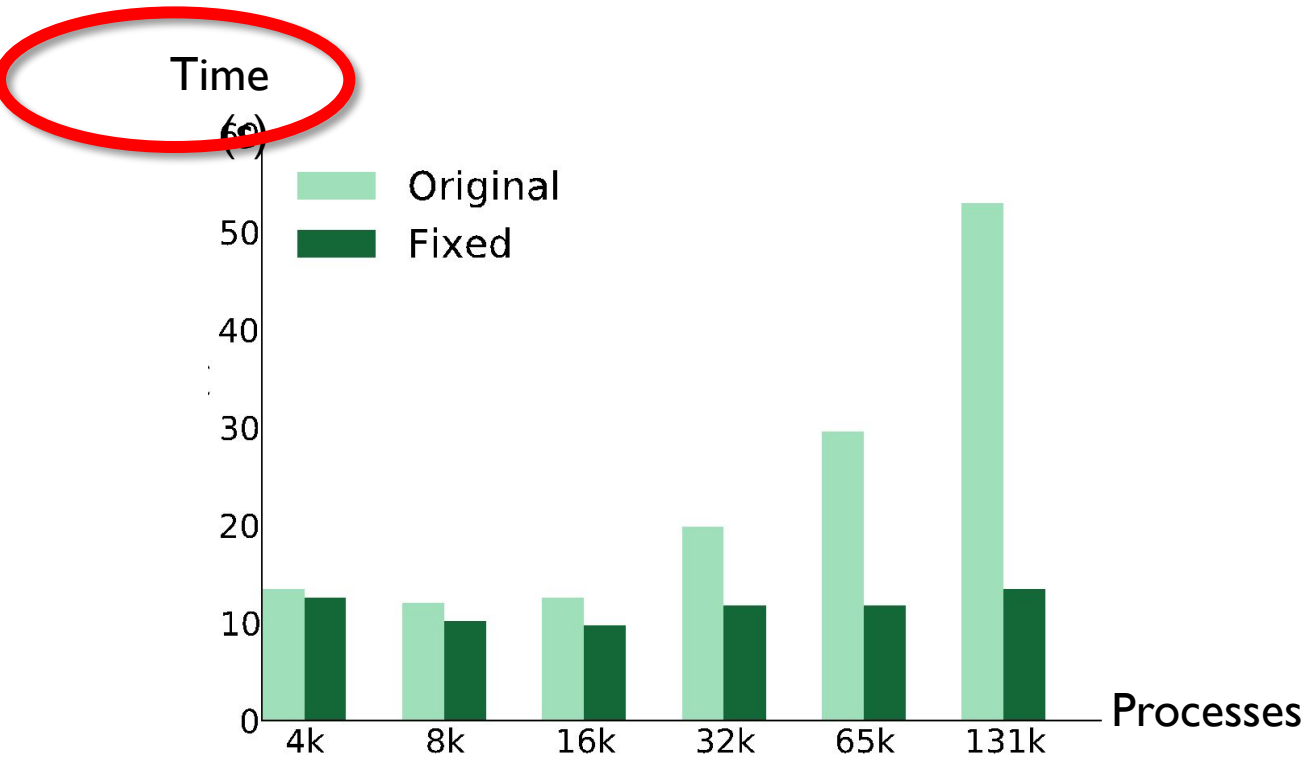
plt.xticks(ind + width / 2, ('G1', 'G2', 'G3', 'G4', 'G5'))
plt.legend(loc='best')
plt.show()
```



Labels on the y-axis need not be vertical



Labels on the y-axis need not be vertical



```
countries = ['USA', 'GB', 'China', 'Russia', 'Germany']
bronzes = np.array([38, 17, 26, 19, 15])
silvers = np.array([37, 23, 18, 18, 10])
golds = np.array([46, 27, 26, 19, 17])
ind = [x for x, _ in enumerate(countries)]

plt.bar(ind, golds, width=0.8, label='golds', color='gold', bottom=silvers+bronzes)
plt.bar(ind, silvers, width=0.8, label='silvers', color='silver', bottom=bronzes)
plt.bar(ind, bronzes, width=0.8, label='bronzes', color='#CD853F')

plt.xticks(ind, countries)
plt.ylabel("Medals")
plt.xlabel("Countries")
plt.legend(loc="upper right")
plt.title("2012 Olympics Top Scorers")

plt.show()
```

