Data Exploration with Python and Jupyter

Basic usage of the Pandas library to download a dataset, explore its contents, clean up missing or invalid data, filter the data according to different criteria, and plot visualizations of the data.

- Part 1: Python and Jupyter
- **Part 2: Pandas with toy data**
- Part 3: Pandas with real data

Press `Spacebar` to go to the next slide (or `?` to see all navigation shortcuts)

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

Pandas

is a data analysis and manipulation Python library

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

Pandas

is a data analysis and manipulation Python library

In [1]:
```
# Import the Pandas library
import pandas as pd
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

Pandas

is a data analysis and manipulation Python library

In [1]:

```
# Import the Pandas library
import pandas as pd
```

In [2]:

```
# Import some toy data as a pandas DataFrame
df = pd.read_csv("https://ssciwr.github.io/jupyter-data-exploration/data
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

Pandas

is a data analysis and manipulation Python library

In [1]:
```
# Import the Pandas library
import pandas as pd
```

In [2]:
```
# Import some toy data as a pandas DataFrame
df = pd.read_csv("https://ssciwr.github.io/jupyter-data-exploration/data
```

In [3]:
```
type(df)
```

Out[3]:
```
pandas.core.frame.DataFrame
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

Pandas

is a data analysis and manipulation Python library

In [1]:
```
# Import the Pandas library
import pandas as pd
```

In [2]:
```
# Import some toy data as a pandas DataFrame
df = pd.read_csv("https://ssciwr.github.io/jupyter-data-exploration/data
```

In [3]:
```
type(df)
```

Out[3]:
```
pandas.core.frame.DataFrame
```

In [4]:
```
len(df)
```

Out[4]:
```
20
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [5]:

```
# Display the first few rows of data
df.head()
```

Out[5]:

	Name	Age	Sex	Height	Eye colour	Wears glasses
0	Bob	12	Male	130.0	blue	yes
1	Simon	13	Male	120.0	blue	no
2	Clare	15	Female	142.5	green	no
3	Jose	11	Male	117.0	brown	no
4	Hannah	9	Female	111.0	blue	yes

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [6]:

```python
# Display general DataFrame info (columns, entries, types)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20 entries, 0 to 19
Data columns (total 6 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   Name            20 non-null     object 
 1   Age             20 non-null     int64  
 2   Sex             20 non-null     object 
 3   Height          20 non-null     float64
 4   Eye colour      20 non-null     object 
 5   Wears glasses   20 non-null     object 
dtypes: float64(1), int64(1), object(4)
memory usage: 1.1+ KB
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

Selecting rows and columns

Three main ways of doing this:

- Python-style indexing operator `[]`
- Pandas `loc` function (label-based)
- Pandas `iloc` function (index-based)

We'll start with the more intuitive Python-style methods, and later move into the more powerful `loc` and `iloc` alternatives

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [7]:

```
# A DataFrame is a bit like a Dictionary - we can lookup columns by name
names = df["Name"]
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [7]:
```
# A DataFrame is a bit like a Dictionary - we can lookup columns by name
names = df["Name"]
```

In [8]:
```
# A column of a DataFrame is a Series
type(names)
```

Out[8]:
```
pandas.core.series.Series
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [7]:
```
# A DataFrame is a bit like a Dictionary - we can lookup columns by name
names = df["Name"]
```

In [8]:
```
# A column of a DataFrame is a Series
type(names)
```

Out[8]:
```
pandas.core.series.Series
```

In [9]:
```
names.head()
```

Out[9]:
```
0       Bob
1     Simon
2     Clare
3      Jose
4    Hannah
Name: Name, dtype: object
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [10]:

```
# A Series is a bit like a List - we can select items by index
names[0]
```

Out[10]:

```
'Bob'
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [10]:

```
# A Series is a bit like a List - we can select items by index
names[0]
```

Out[10]:

```
'Bob'
```

In [11]:

```
# Here are the first three items:
names[0:3]
```

Out[11]:

```
0      Bob
1    Simon
2    Clare
Name: Name, dtype: object
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [10]:
```
# A Series is a bit like a List - we can select items by index
names[0]
```

Out[10]:
```
'Bob'
```

In [11]:
```
# Here are the first three items:
names[0:3]
```

Out[11]:
```
0      Bob
1    Simon
2    Clare
Name: Name, dtype: object
```

In [12]:
```
# Can also iterate over items
for name in names:
    print(name, "", end="")
```

```
Bob Simon Clare Jose Hannah Ryan Craig Suzy Chris Josie Claire J
ohn Agnes Robert Julia Fabian Joseph Roberta Chris Lucas
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [13]:

```
# alternative syntax: dataframe.column_name
# both of these are equivalent:
ages1 = df["Age"]
print(ages1.head())

ages2 = df.Age
print(ages2.head())

# note: this only works if the column label is a valid python object nam
```

```
0    12
1    13
2    15
3    11
4     9
Name: Age, dtype: int64
0    12
1    13
2    15
3    11
4     9
Name: Age, dtype: int64
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

iloc

- select data based on its *location*
- first specify row(s), then column(s): `df.iloc[row, col]`
- treating dataset as a matrix, or a list of lists
- note: slices are *exclusive*, i.e. `df.iloc[0:2]` returns rows 0 and 1, but not 2

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [14]:

```
# First row of data (column is implicitly "all" if not specified)
df.iloc[0]
```

Out[14]:

```
Name                 Bob
Age                   12
Sex                 Male
Height             130.0
Eye colour          blue
Wears glasses        yes
Name: 0, dtype: object
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [14]:
```
# First row of data (column is implicitly "all" if not specified)
df.iloc[0]
```

Out[14]:
```
Name                   Bob
Age                     12
Sex                   Male
Height               130.0
Eye colour            blue
Wears glasses          yes
Name: 0, dtype: object
```

In [15]:
```
# First row of data (using : slice operator to select all columns)
df.iloc[0, :]
```

Out[15]:
```
Name                   Bob
Age                     12
Sex                   Male
Height               130.0
Eye colour            blue
Wears glasses          yes
Name: 0, dtype: object
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [16]:
```
# First column of data
df.iloc[:, 0].head()
```

Out[16]:
```
0       Bob
1     Simon
2     Clare
3      Jose
4    Hannah
Name: Name, dtype: object
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [16]:
```python
# First column of data
df.iloc[:, 0].head()
```

Out[16]:
```
0       Bob
1     Simon
2     Clare
3      Jose
4    Hannah
Name: Name, dtype: object
```

In [17]:
```python
# Can select slices of rows and columns: e.g. first 3 rows, last 2 colum
df.iloc[0:3, -2:]
```

Out[17]:

	Eye colour	Wears glasses
0	blue	yes
1	blue	no
2	green	no

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [16]:
```
# First column of data
df.iloc[:, 0].head()
```
Out[16]:
```
0        Bob
1      Simon
2      Clare
3       Jose
4     Hannah
Name: Name, dtype: object
```
In [17]:
```
# Can select slices of rows and columns: e.g. first 3 rows, last 2 colum
df.iloc[0:3, -2:]
```
Out[17]:

	Eye colour	Wears glasses
0	blue	yes
1	blue	no
2	green	no

In [18]:
```
# Can also select a list of indices, e.g. rows 3,5,7, columns 3,5
df.iloc[[3, 5, 7], [3, 5]]
```
Out[18]:

	Height	Wears glasses
3	117.0	no
5	124.0	no
7	137.0	yes

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

loc

- select data based on its index *label* and column *label*, instead of location
- first specify row(s), then column(s): `df.loc[:, "Name"]`
- often the most useful method
- note: slices are *inclusive*, i.e. `df.loc[0:2]` returns rows 0 and 1 *and* 2
- note: in our example, the index label is a number, and it is the same as the row index, but in general this is not the case

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [19]:

```
# Row with index label "0" (column is implicitly "all" if not specified)
df.loc[0]
```

Out[19]:

```
Name                   Bob
Age                     12
Sex                   Male
Height               130.0
Eye colour            blue
Wears glasses          yes
Name: 0, dtype: object
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [19]:
```
# Row with index label "0" (column is implicitly "all" if not specified)
df.loc[0]
```

Out[19]:
```
Name                   Bob
Age                     12
Sex                   Male
Height               130.0
Eye colour            blue
Wears glasses          yes
Name: 0, dtype: object
```

In [20]:
```
# Row with index label "0" (using : slice operator to select all columns
df.loc[0, :]
```

Out[20]:
```
Name                   Bob
Age                     12
Sex                   Male
Height               130.0
Eye colour            blue
Wears glasses          yes
Name: 0, dtype: object
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [21]:

```
# "Name" column of data (using : slice operator to select all rows)
df.loc[:, "Name"].head()
```

Out[21]:

```
0       Bob
1     Simon
2     Clare
3      Jose
4    Hannah
Name: Name, dtype: object
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [21]:

```
# "Name" column of data (using : slice operator to select all rows)
df.loc[:, "Name"].head()
```

Out[21]:

```
0       Bob
1     Simon
2     Clare
3      Jose
4    Hannah
Name: Name, dtype: object
```

In [22]:

```
# Can also select a list of labels, e.g. index labels 3,5,7, columns "He
df.loc[[3, 5, 7], ["Height", "Wears glasses"]]
```

Out[22]:

	Height	Wears glasses
3	117.0	no
5	124.0	no
7	137.0	yes

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

Conditionals

- a statement that is either true or false
    - `a == b` : true if `a` is equal to `b`
    - `a != b` : true if `a` is not equal to `b`
    - `a > b` : true if `a` is greater than `b`
    - `a >= b` : true if `a` is greater than or equal to `b`
    - `a < b` : true if `a` is less than `b`
    - `a <= b` : true if `a` is less than or equal to `b`
- they can be combined
    - `a & b` : true if a and b are both true, otherwise false
    - `a | b` : true if a or b is true, otherwise false
- if `a` is a pandas Series, the result is a Boolean Series
    - with a True or False result for each row
    - which can be used by loc to select data
- this is very flexible and powerful

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [23]:

```
# This returns True, as the condition 10 > 9 is true
10 > 9
```

Out[23]:

```
True
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [24]:

```python
# Similarly, this returns False, as the condition 8 > 9 is false
8 > 9
```

Out[24]:

```
False
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [25]:

```python
# Can do the same with a Series - returns a Boolean (true/false) Series
df["Age"] > 9
```

Out[25]:

```
0      True
1      True
2      True
3      True
4     False
5      True
6      True
7      True
8      True
9     False
10     True
11    False
12    False
13    False
14    False
15    False
16    False
17     True
18    False
19    False
Name: Age, dtype: bool
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [26]:

```python
# loc can take this as the selection, e.g. older than 9
df.loc[df["Age"] > 9]
```

Out[26]:

	Name	Age	Sex	Height	Eye colour	Wears glasses
0	Bob	12	Male	130.0	blue	yes
1	Simon	13	Male	120.0	blue	no
2	Clare	15	Female	142.5	green	no
3	Jose	11	Male	117.0	brown	no
5	Ryan	11	Male	124.0	brown	no
6	Craig	12	Male	124.0	brown	no
7	Suzy	14	Female	137.0	grey	yes
8	Chris	10	Male	112.5	brown	no
10	Claire	16	Female	158.0	blue	no
17	Roberta	11	Female	121.0	grey	no

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [27]:

```
# can combine conditions with & e.g. older than 9 and have blue eyes
df.loc[(df["Age"] > 9) & (df["Eye colour"] == "blue")]
# note: good idea to wrap each condition in brackets when combining them
```

Out[27]:

	Name	Age	Sex	Height	Eye colour	Wears glasses
0	Bob	12	Male	130.0	blue	yes
1	Simon	13	Male	120.0	blue	no
10	Claire	16	Female	158.0	blue	no

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [28]:

```
# can have multiple conditions with | e.g. younger than 7 or wears glass
df.loc[(df["Age"] < 7) | (df["Wears glasses"] == "yes")]
```

Out[28]:

	Name	Age	Sex	Height	Eye colour	Wears glasses
0	Bob	12	Male	130.0	blue	yes
4	Hannah	9	Female	111.0	blue	yes
7	Suzy	14	Female	137.0	grey	yes
11	John	5	Male	97.0	grey	no
12	Agnes	6	Female	98.0	blue	yes
14	Julia	3	Female	63.0	blue	no
15	Fabian	5	Male	101.0	blue	no
16	Joseph	8	Male	107.0	brown	yes
19	Lucas	2	Male	59.0	blue	no

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

Summarizing data

Some useful functions for getting a quick overview of a Series:

- `describe()`
  - for numerical data: mean, min, max, std deviation, etc
  - for strings: count, number of unique items, most common item
- `count()` - the number of items in a Series
- `unique()` - a list of the unique items in a Series
- `value_counts()` - the count of each unique item in a Series
- `plot()` - plot numerical data on the y-axis (with the Index on the x-axis)
- `hist()` - plot a histogram of frequency of each unique item

You can also use these methods on the whole DataFrame, but only numerical data columns will be considered.

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [29]:
```
df["Eye colour"].describe()
```

Out[29]:
```
count       20
unique       4
top       blue
freq         8
Name: Eye colour, dtype: object
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [29]:
```
df["Eye colour"].describe()
```
Out[29]:
```
count       20
unique       4
top       blue
freq         8
Name: Eye colour, dtype: object
```
In [30]:
```
df["Eye colour"].count()
```
Out[30]:
```
20
```
Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [29]:
```
df["Eye colour"].describe()
```

Out[29]:
```
count       20
unique       4
top       blue
freq         8
Name: Eye colour, dtype: object
```

In [30]:
```
df["Eye colour"].count()
```

Out[30]:
```
20
```

In [31]:
```
df["Eye colour"].unique()
```

Out[31]:
```
array(['blue', 'green', 'brown', 'grey'], dtype=object)
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [32]:

```
df["Eye colour"].value_counts()
```

Out[32]:

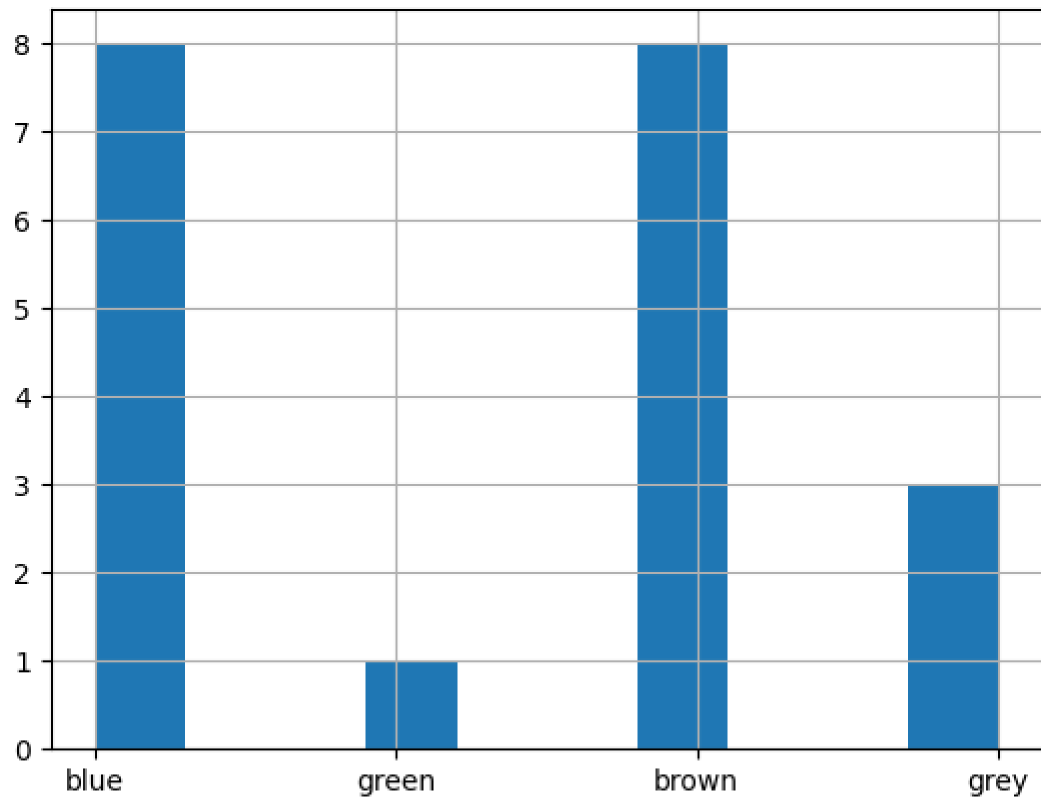
```
Eye colour
blue     8
brown    8
grey     3
green    1
Name: count, dtype: int64
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [33]:
```
df["Eye colour"].hist()
```

Out[33]:

<Axes: >



Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

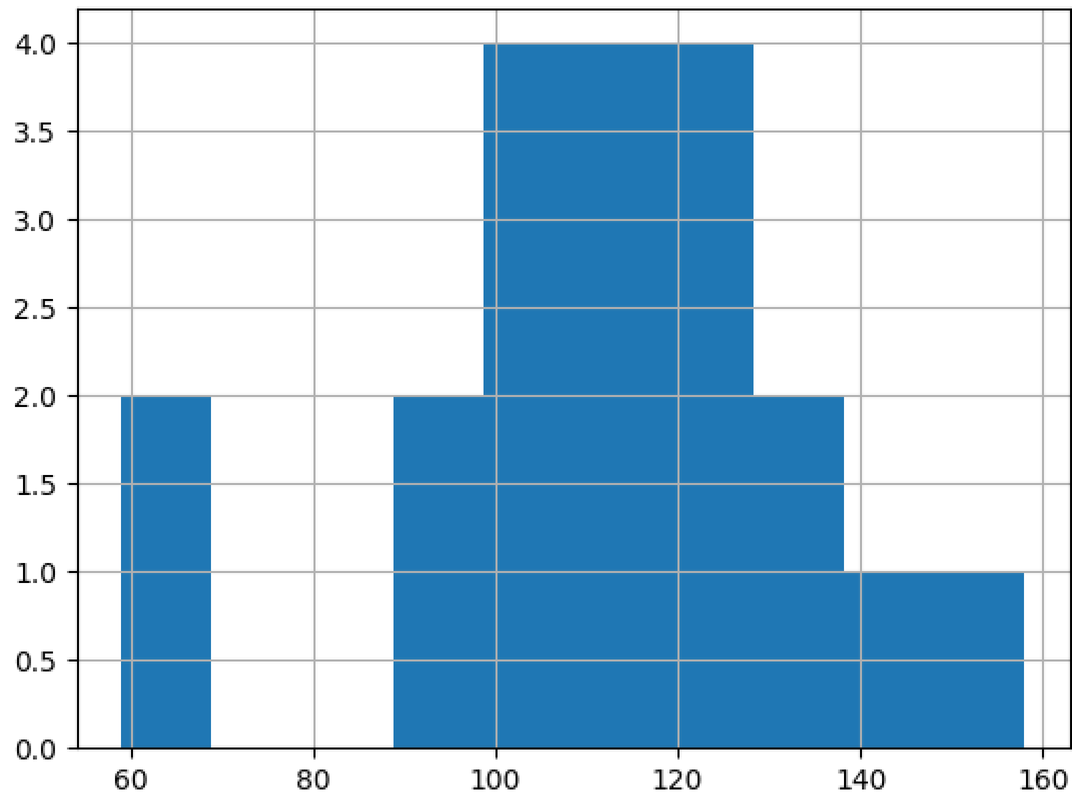
In [34]:
```
df["Height"].describe()
```

Out[34]:
```
count     20.000000
mean     112.200000
std       23.283945
min       59.000000
25%      104.750000
50%      111.750000
75%      124.000000
max      158.000000
Name: Height, dtype: float64
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [35]:
```
df["Height"].hist()
```

Out[35]: <Axes: >



Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

Plotting

- `df.plot` contains various plot methods
    - type `df.plot.` in a code cell then press `Tab` to see them listed
    - or read the docs by typing `?df.plot` in a code cell and running the cell
- specify the column to plot with `x="Column Name"`
- commonly used plots
    - `line` for time series data
    - `hist` for categorical data
    - `scatter` to plot the relationship between two columns
- can use `plot` method on Series or Dataframe
- returns a Matplotlib object

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

Matplotlib

- terminology
  - `figure`: the "canvas" on which plots will be made
  - `axis`: a plot - a figure can have one or several of these
  - these are implicitly created if you don't do so yourself
- useful commands
  - `ax = plt.subplot()`: create an axis you can pass to Pandas to plot on
  - `fig, axs = plt.subplots(ncols=3)`: create a figure and multiple axes you can pass to Pandas to plot on
  - `plt.savefig("plot.png")`: save the plot as an image

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [36]:
```
# import matplotlib

import matplotlib.pyplot as plt
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [36]:

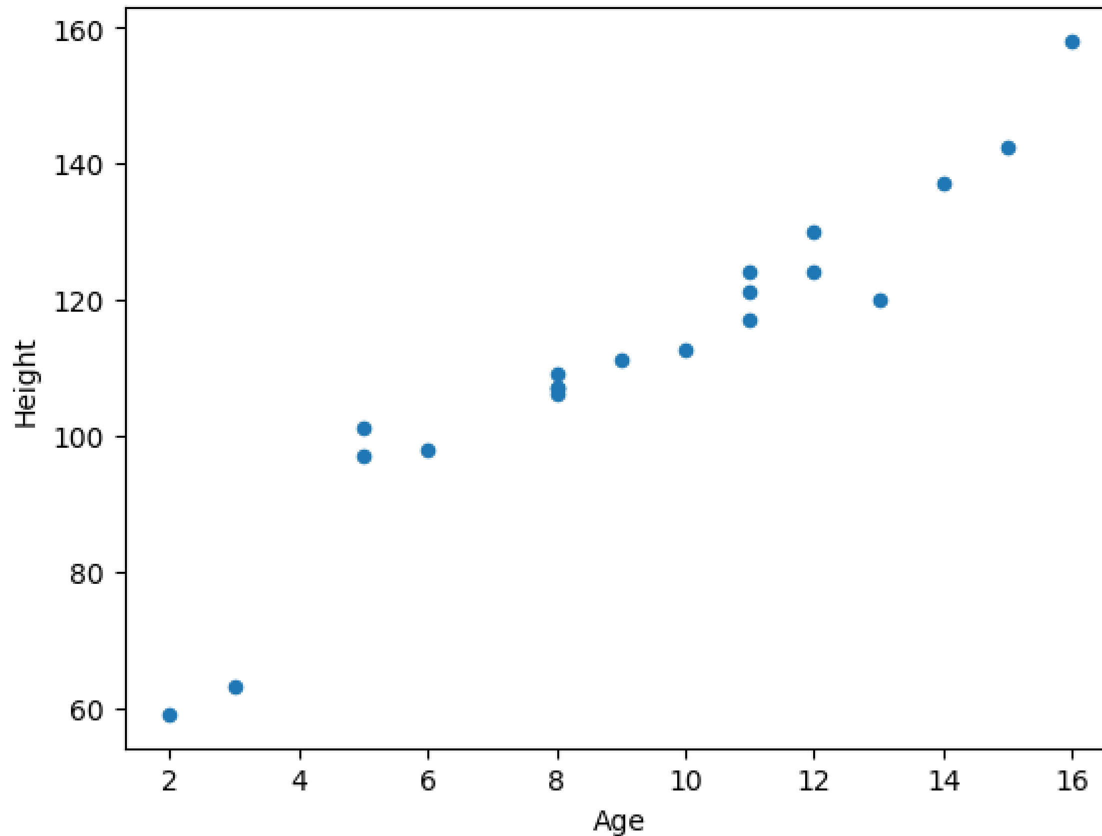
```
# import matplotlib

import matplotlib.pyplot as plt
```

In [37]:

```
# see how two columns are correlated with a scatter plot
df.plot.scatter(x="Age", y="Height")
```

Out[37]:

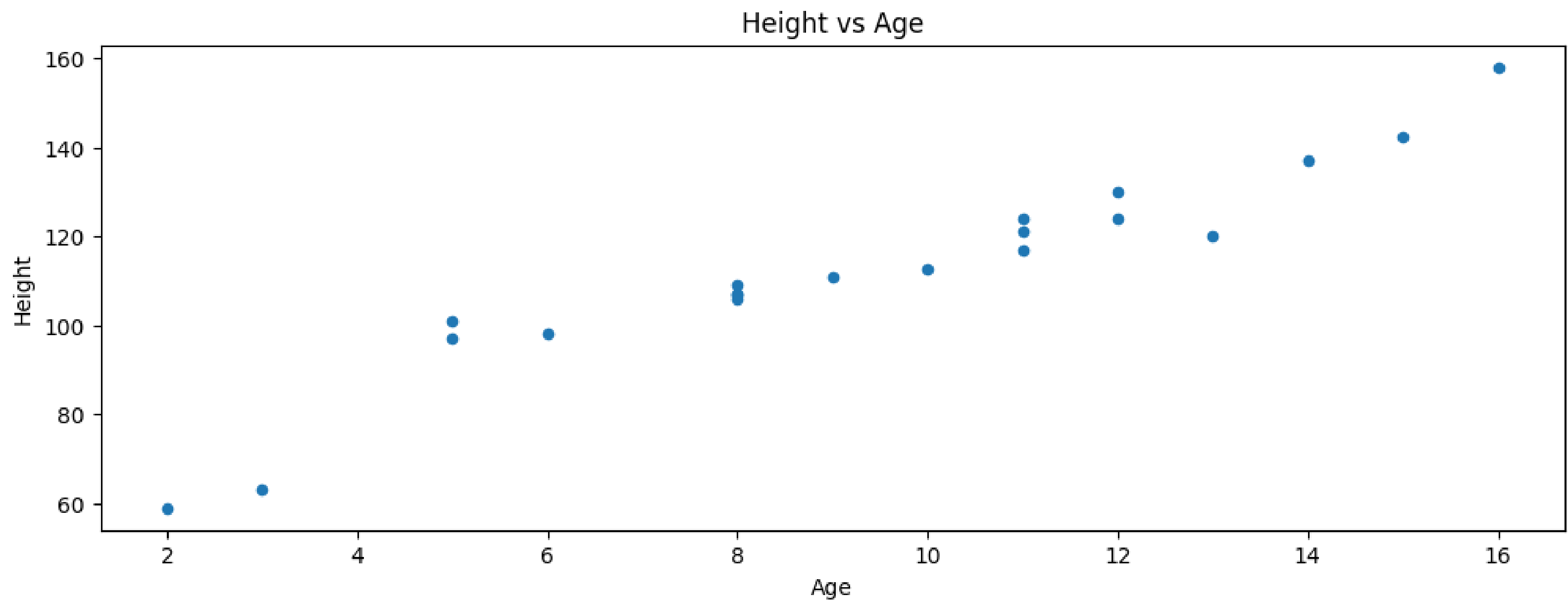
```
<Axes: xlabel='Age', ylabel='Height'>
```



Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [38]:

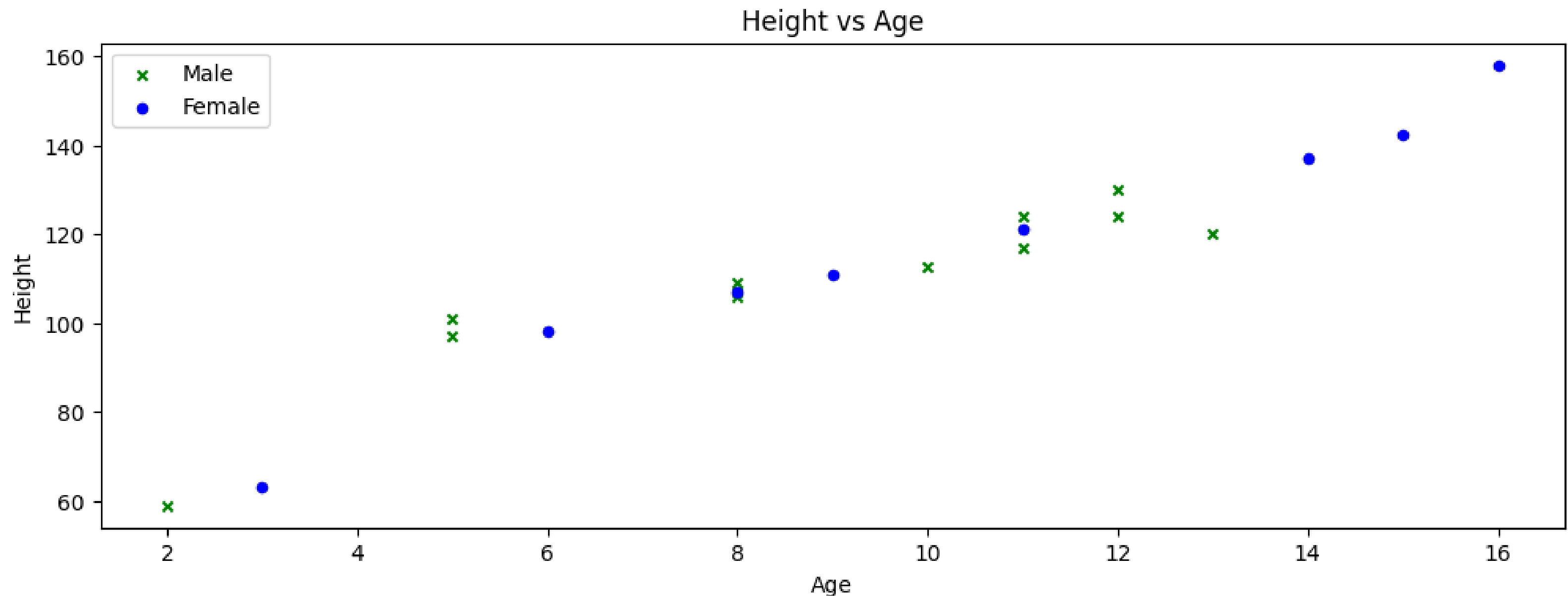
```
# do the same thing, but use matplotlib to customise the plot
# make a larger figure
fig, axs = plt.subplots(figsize=(12, 4))
# pass our axis to pandas plot
df.plot.scatter(x="Age", y="Height", ax=axs)
# set a title
plt.title("Height vs Age")
# display the plot
plt.show()
```



Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [39]:

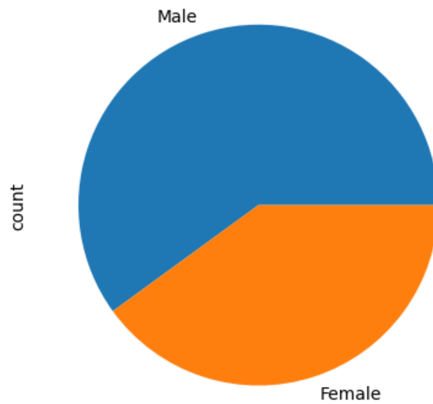
```
# filter the data before plotting, and plot multiple labelled datapoints
fig, axs = plt.subplots(figsize=(12, 4))
df.loc[df["Sex"] == "Male"].plot.scatter(
    x="Age", y="Height", ax=axs, label="Male", marker="x", color="green'
)
df.loc[df["Sex"] == "Female"].plot.scatter(
    x="Age", y="Height", ax=axs, label="Female", marker="o", color="blue
)
plt.legend()
plt.title("Height vs Age")
plt.show()
```

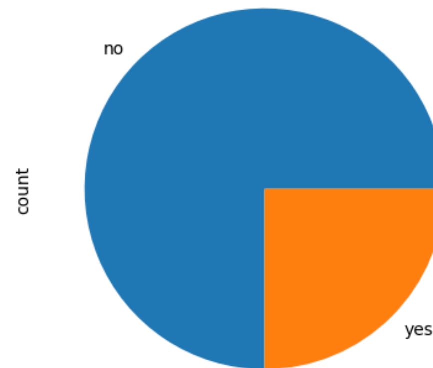


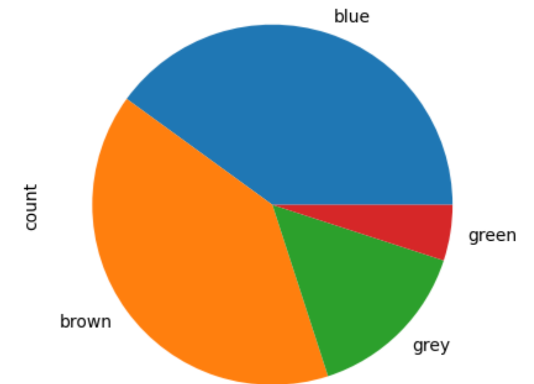
Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [40]:
```
fig, axs = plt.subplots(nrows=1, ncols=3, figsize=(16, 6))
df["Sex"].value_counts().plot.pie(ax=axs[0])
df["Wears glasses"].value_counts().plot.pie(ax=axs[1])
df["Eye colour"].value_counts().plot.pie(ax=axs[2])
plt.plot()
```

Out[40]:
```
[]
```







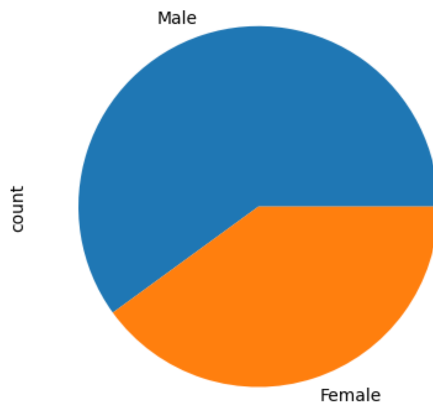
Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

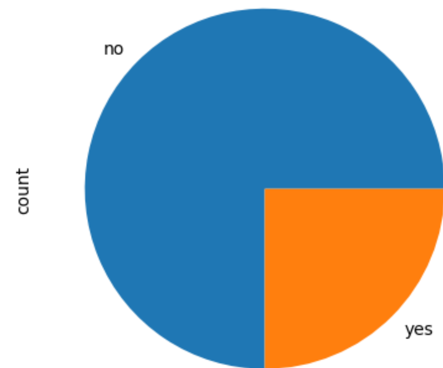
In [41]:

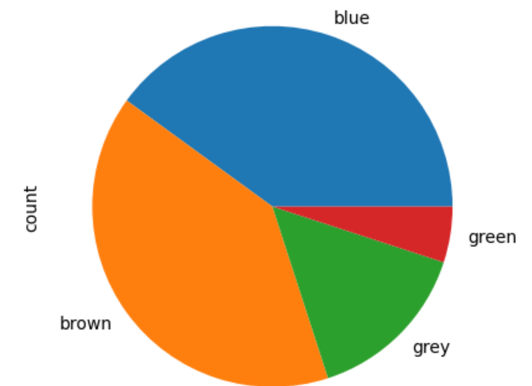
```
fig, axs = plt.subplots(nrows=1, ncols=3, figsize=(16, 6))
for ax, column in zip(axs, ["Sex", "Wears glasses", "Eye colour"]):
    df[column].value_counts().plot.pie(ax=ax)
plt.plot()
```

Out[41]:

```
[]
```







Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

Groupby

- **split** the data into groups
- **apply** some function to each group
- **combine** the results

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [42]:

```
grouped = df.groupby(["Sex"])
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [42]:
```
grouped = df.groupby(["Sex"])
```

In [43]:
```
type(grouped)
```

Out[43]:
```
pandas.core.groupby.generic.DataFrameGroupBy
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [42]:
```
grouped = df.groupby(["Sex"])
```

In [43]:
```
type(grouped)
```

Out[43]:
```
pandas.core.groupby.generic.DataFrameGroupBy
```

In [44]:
```
grouped.groups
```

Out[44]:
```
{'Female': [2, 4, 7, 9, 10, 12, 14, 17], 'Male': [0, 1, 3, 5,
6, 8, 11, 13, 15, 16, 18, 19]}
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [45]:

```
for group, data in grouped:
    print(group)
    print(data.head())
```

```
('Female',)
      Name  Age     Sex  Height Eye colour Wears glasses
2    Clare   15  Female   142.5      green            no
4   Hannah    9  Female   111.0       blue           yes
7     Suzy   14  Female   137.0       grey           yes
9    Josie    8  Female   107.0      brown            no
10  Claire   16  Female   158.0       blue            no
('Male',)
    Name  Age   Sex  Height Eye colour Wears glasses
0    Bob   12  Male   130.0       blue           yes
1  Simon   13  Male   120.0       blue            no
3   Jose   11  Male   117.0      brown            no
5   Ryan   11  Male   124.0      brown            no
6  Craig   12  Male   124.0      brown            no
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [46]:

```
df.groupby(["Sex"])["Age"].count()
```

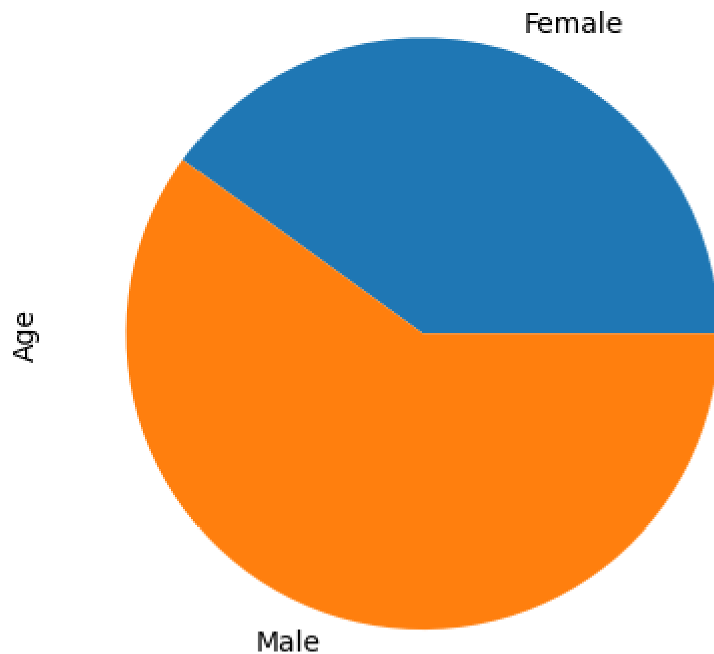
Out[46]:

```
Sex
Female     8
Male      12
Name: Age, dtype: int64
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [47]:
```python
df.groupby(["Sex"])["Age"].count().plot.pie()
```

Out[47]:
```
<Axes: ylabel='Age'>
```



Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [48]:

```
# equivalent "by hand" version
index = []
data = []
# split by sex
for group in df.Sex.unique():
    # apply "count" function to each subset
    count = df.loc[df.Sex == group].Age.count()
    # combine results back into a series
    index.append(group)
    data.append(count)
count_by_age = pd.Series(index=index, data=data, name="Age")
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [48]:

```
# equivalent "by hand" version
index = []
data = []
# split by sex
for group in df.Sex.unique():
    # apply "count" function to each subset
    count = df.loc[df.Sex == group].Age.count()
    # combine results back into a series
    index.append(group)
    data.append(count)
count_by_age = pd.Series(index=index, data=data, name="Age")
```

In [49]:

```
count_by_age
```

Out[49]:

```
Male      12
Female     8
Name: Age, dtype: int64
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [50]:

```
# can group-by multiple columns
multigroup = df.groupby(["Eye colour", "Sex"]).Name.count()
multigroup
```

Out[50]:

```
Eye colour  Sex
blue        Female    4
            Male      4
brown       Female    1
            Male      7
green       Female    1
grey        Female    2
            Male      1
Name: Name, dtype: int64
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [51]:

```python
# for plotting, unstack is helpful: converts our multi-index series intc
multigroup.unstack()
```

Out[51]:

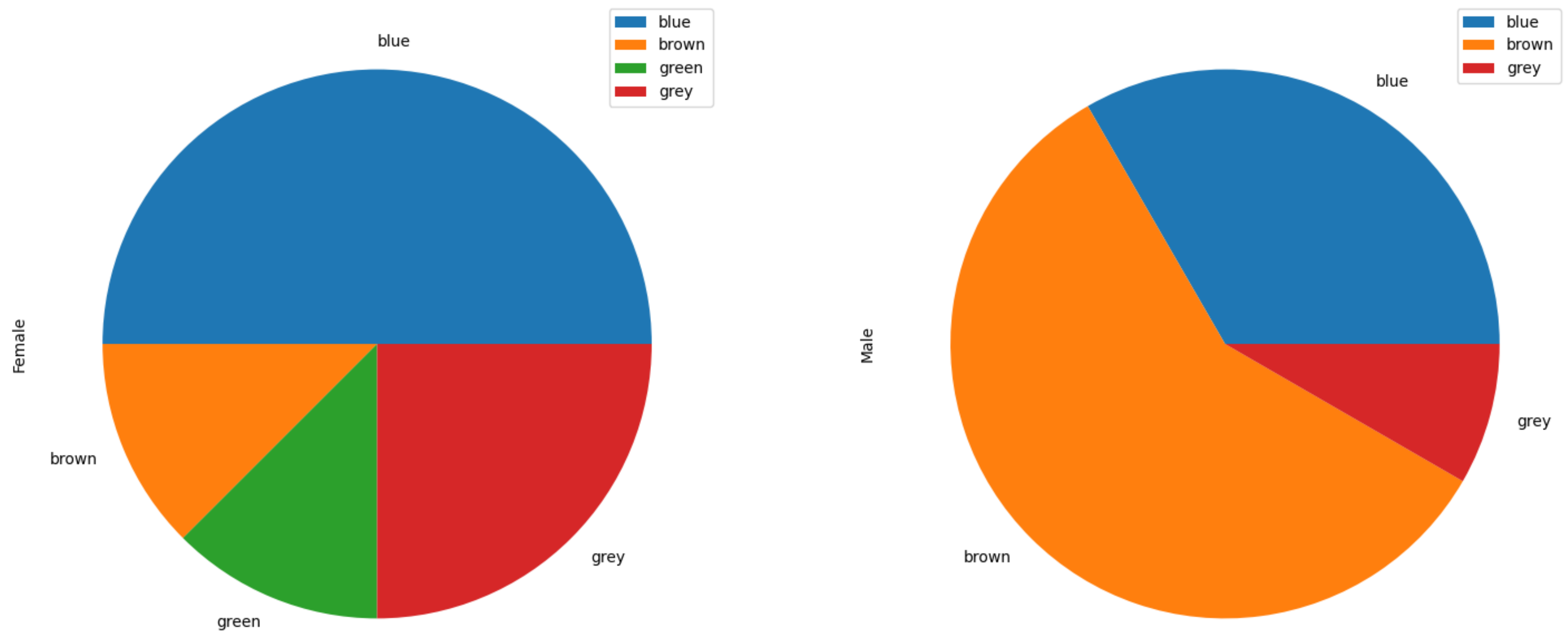
Sex	Female	Male
Eye colour		
blue	4.0	4.0
brown	1.0	7.0
green	1.0	NaN
grey	2.0	1.0

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [52]:
```
multigroup.unstack().plot.pie(subplots=True, figsize=(18, 8))
```

Out[52]:

```
array([<Axes: ylabel='Female'>, <Axes: ylabel='Male'>], dtype=o
bject)
```



Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [53]:

```
# custom apply function: count people older than 10 years old
def count_older_than_ten(series):
    return series.loc[series > 10].count()


df.groupby(["Sex"])["Age"].apply(count_older_than_ten)
```

Out[53]:

```
Sex
Female    4
Male      5
Name: Age, dtype: int64
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [53]:
```
# custom apply function: count people older than 10 years old
def count_older_than_ten(series):
    return series.loc[series > 10].count()


df.groupby(["Sex"])["Age"].apply(count_older_than_ten)
```

Out[53]:
```
Sex
Female    4
Male      5
Name: Age, dtype: int64
```

In [54]:
```
# same thing with a lambda instead of defining a function
df.groupby(["Sex"])["Age"].apply(lambda x: x.loc[x > 10].count())
```

Out[54]:
```
Sex
Female    4
Male      5
Name: Age, dtype: int64
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

Types

- Default type is `Object`, aka String
- Pandas tries to identify types like numbers, dates and booleans
- Can also tell Pandas what type a column should be
- Using the correct type has multiple benefits
  - better performance (use less RAM, faster to run)
  - more functionality (e.g. summary / plots of numerical types)

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [55]:
```python
# display type of each column
df.dtypes
```

Out[55]:
```
Name               object
Age                 int64
Sex                object
Height            float64
Eye colour         object
Wears glasses      object
dtype: object
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [55]:
```
# display type of each column
df.dtypes
```

Out[55]:
```
Name              object
Age                int64
Sex               object
Height           float64
Eye colour        object
Wears glasses     object
dtype: object
```

In [56]:
```
# display memory usage of each column
df.memory_usage(deep=True)
```

Out[56]:
```
Index             128
Name             1241
Age               160
Sex              1236
Height            160
Eye colour       1229
Wears glasses    1185
dtype: int64
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [57]:
```
# list unique values in "Sex" column:
df["Sex"].unique()
```
Out[57]:
```
array(['Male', 'Female'], dtype=object)
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [57]:
```
# list unique values in "Sex" column:
df["Sex"].unique()
```

Out[57]:
```
array(['Male', 'Female'], dtype=object)
```

In [58]:
```
# see how much RAM is used to store this column as strings
df["Sex"].memory_usage(deep=True)
```

Out[58]:
```
1364
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [59]:

```
# convert to a category type
df["Sex"] = df["Sex"].astype("category")
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [59]:

```
# convert to a category type
df["Sex"] = df["Sex"].astype("category")
```

In [60]:

```
# list unique values in column:
df["Sex"].unique()
```

Out[60]:

```
['Male', 'Female']
Categories (2, object): ['Female', 'Male']
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [59]:
```
# convert to a category type
df["Sex"] = df["Sex"].astype("category")
```

In [60]:
```
# list unique values in column:
df["Sex"].unique()
```

Out[60]:
```
['Male', 'Female']
Categories (2, object): ['Female', 'Male']
```

In [61]:
```
# check RAM usage now
df["Sex"].memory_usage(deep=True)
```

Out[61]:
```
380
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [59]:
```
# convert to a category type
df["Sex"] = df["Sex"].astype("category")
```

In [60]:
```
# list unique values in column:
df["Sex"].unique()
```

Out[60]:
```
['Male', 'Female']
Categories (2, object): ['Female', 'Male']
```

In [61]:
```
# check RAM usage now
df["Sex"].memory_usage(deep=True)
```

Out[61]:
```
380
```

In [62]:
```
# do the same for eye colour
df["Eye colour"] = df["Eye colour"].astype("category")
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [63]:
```
# list unique values to confirm Wears glasses is really a boolean:
df["Wears glasses"].unique()
```

Out[63]:
```
array(['yes', 'no'], dtype=object)
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [63]:
```
# list unique values to confirm Wears glasses is really a boolean:
df["Wears glasses"].unique()
```

Out[63]:
```
array(['yes', 'no'], dtype=object)
```

In [64]:
```
# see how much RAM is used to store this column as strings
df["Wears glasses"].memory_usage(deep=True)
```

Out[64]:
```
1313
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [63]:
```
# list unique values to confirm Wears glasses is really a boolean:
df["Wears glasses"].unique()
```
Out[63]:
```
array(['yes', 'no'], dtype=object)
```
In [64]:
```
# see how much RAM is used to store this column as strings
df["Wears glasses"].memory_usage(deep=True)
```
Out[64]:
```
1313
```
In [65]:
```
# convert "yes" to True, "no" to False
df["Wears glasses"] = df["Wears glasses"].map({"yes": True, "no": False}
df["Wears glasses"].unique()
```
Out[65]:
```
array([ True, False])
```
Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [63]:
```
# list unique values to confirm Wears glasses is really a boolean:
df["Wears glasses"].unique()
```

Out[63]:
```
array(['yes', 'no'], dtype=object)
```

In [64]:
```
# see how much RAM is used to store this column as strings
df["Wears glasses"].memory_usage(deep=True)
```

Out[64]:
```
1313
```

In [65]:
```
# convert "yes" to True, "no" to False
df["Wears glasses"] = df["Wears glasses"].map({"yes": True, "no": False}
df["Wears glasses"].unique()
```

Out[65]:
```
array([ True, False])
```

In [66]:
```
df["Wears glasses"].memory_usage(deep=True)
```

Out[66]:
```
148
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

Next

- Part 3: Pandas with real data

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js