Errors and Exception Handling

In [2]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
import statsmodels as sm
```

In [4]:
```
float('something')
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-4-2649e4ade0e6> in <module>()
----> 1 float('something')

ValueError: could not convert string to float: 'something'
```

try/except block

In [15]:
```
def attempt_float(x):
    try:
        return float(x)
    except:
        print ("Bad Input")
        return x
```

The code in the except part of the block will only be executed if float(x) raises an exception:

In [16]:
```
attempt_float('1.2345')
```

Out[16]:
```
1.2345
```

In [17]:
```
attempt_float('something')
```

```
Bad Input
```

Out[17]:
```
'something'
```

You might want to only suppress ValueError, since a TypeError (the input was not a string or numeric value) might indicate a legitimate bug in your program. To do that, write the exception type after except:

In [18]:
```
def attempt_float(x):
    try:
        return float(x)
    except ValueError:
        print ("Bad Input")
        return x
```

In [19]:
```
attempt_float((1, 2))
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-19-8b0026e9e6b7> in <module>()
----> 1 attempt_float((1, 2))

<ipython-input-18-b037433328fa> in attempt_float(x)
      1 def attempt_float(x):
      2     try:
----> 3         return float(x)
      4     except ValueError:
      5         print ("Bad Input")

TypeError: float() argument must be a string or a number, not 'tuple'
```

You can catch multiple exception types by writing a tuple of exception types instead (the parentheses are required):

In [21]:
```
def attempt_float(x):
    try:
        return float(x)
    except (TypeError, ValueError):
        return x
```

In some cases, you may not want to suppress an exception, but you want some code to be executed regardless of whether the code in the try block succeeds or not. To do this, use **finally**:

In [23]:
```
f = open(path, 'w')
try:
    write_to_file(f)
finally:
    f.close() #Here, the file handle f will always get closed.
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-23-b2dd13849a39> in <module>()
----> 1 f = open(path, 'w')
      2 try:
      3     write_to_file(f)
      4 finally:
      5     f.close() #Here, the file handle f will always get closed.

NameError: name 'path' is not defined
```

Similarly, you can have code that executes only if the try: block succeeds using **else**:

In [26]:
```
f = open(path, 'w')
try:
    write_to_file(f)
except:
    print('Failed')
else:
    print('Succeeded')
finally:
    f.close()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-26-9e425aa0e563> in <module>()
----> 1 f = open(path, 'w')
        global f = undefined
        global open = undefined
        global path = undefined
      2 try:
      3     write_to_file(f)
      4 except:
      5     print('Failed')

NameError: name 'path' is not defined
```

Exceptions in IPython and Jupyter

Having additional context by itself is a big advantage over the standard Python inter- preter (which does not provide any additional context). You can control the amount of context shown using the **%xmode** magic command, from Plain (same as the stan- dard Python interpreter) to Verbose (which inlines function argument values and more). You can step into the stack (using the **%debug%** or **%pdb** magics) after an error has occurred for interactive post-mortem debugging.

An **assert** statement will check to make sure that something is true during the course of a program. If the condition if false, the program stops.

In [1]:
```
a = 5
```

In [2]:
```
assert (a > 6)
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
<ipython-input-2-ec67f926a25a> in <module>()
----> 1 assert (a > 6)

AssertionError:
```

In [3]:
```
assert (a <6)
```

In [ ]:
```

```