The official project homepage Goal Extend what we learned about Series objects in the previous tutorial to their 2D counterpart -**DataFrames** Develop some tools for dealing with missing data (not exhaustive, but a start) **DataFrames** Pandas quick start guide for DataFrames A DataFrame (DF) is a labeled data struture that can be thought of as a 2D extension of the Series objects that we discussed in the first part of the tutorial A DF can accept many types of input, multiple Series, a dict of 1D arrays, another DF, etc Like a Series, DFs contain data values and their labels. Because we're now dealing with a 2D structure, we call the row labels the index argument and the column labels the column argument. Like a Series, if you don't explicitly assign row and column labels, then they will be autogenerated (but not as useful as specifying the labels yourself!) Much of what we learned about Series objects will generalize to DFs, so here we'll focus on some of key functionality that might not be obvious based on the first part of the tutorial. One more quick note: if using an older version of Python (earlier than 3.6) and Pandas (earlier than 0.23) and you create a DF from a dict without explicitly specifying column names, then the column names will be entered into the DF based on lexical order Import libs # import a generic pandas object and also a few specific functions that we'll use import pandas as pd from google.colab import files Upload a file to the /content folder on google colab Select the file you want to upload (the csv file that I sent out) It will load into your 'contents' folder Then you can interact with it just like a normal file on your hardrive %ls annual temp2.csv sample_data/ files.upload() Upload widget is only available when the cell has been executed in Choose Files No file chosen the current browser session. Please rerun this cell to enable. Saving annual temp csv2.csv to annual temp csv2.csv Out[]: {'annual_temp_csv2.csv': b'Source, Year, Mean\r\nGCAG, 2016, 0.9363\r\nGISTEMP, 2016, \r\nGC AG, 2015, \r\nGISTEMP, 2015, 0.87\r\nGCAG, 2014, 0.7408\r\nGISTEMP, 2014, 0.74\r\nGCAG, 2013, 0. 6679\r\nGISTEMP,2013,0.65\r\nGCAG,2012,0.624\r\nGISTEMP,2012,0.63\r\nGCAG,2011,0.5788 \r\nGISTEMP,2011,0.6\r\nGCAG,2010,0.7014\r\nGISTEMP,2010,0.71\r\nGCAG,2009,0.6367\r\nG ISTEMP, 2009, 0.64\r\nGCAG, 2008, 0.5419\r\nGISTEMP, 2008, 0.54\r\nGCAG, 2007, 0.61\r\nGISTEM P,2007,0.66\r\ngCAG,2006,0.6125\r\ngISTEMP,2006,0.63\r\ngCAG,2005,0.6585\r\ngISTEMP,20 05,0.69\r\nGCAG,2004,0.5783\r\nGISTEMP,2004,0.55\r\nGCAG,2003,0.6134\r\nGISTEMP,2003, 0.62\r\nGCAG,2002,0.6023\r\nGISTEMP,2002,0.63\r\nGCAG,2001,0.5473\r\nGISTEMP,2001,0.55 \r\ngCAG,2000,0.4262\r\ngISTEMP,2000,0.42\r\ngCAG,1999,0.4438\r\ngISTEMP,1999,0.42\r\n GCAG, 1998, 0.6344\r\nGISTEMP, 1998, 0.64\r\nGCAG, 1997, 0.5187\r\nGISTEMP, 1997, 0.48\r\nGCA G,1996,0.3228\r\nGISTEMP,1996,0.35\r\nGCAG,1995,0.4577\r\nGISTEMP,1995,0.46\r\nGCAG,19 94,0.3409\r\nGISTEMP,1994,0.32\r\nGCAG,1993,0.2853\r\nGISTEMP,1993,0.24\r\nGCAG,1992, 0.2571\r\nGISTEMP,1992,0.23\r\nGCAG,1991,0.4055\r\nGISTEMP,1991,0.43\r\nGCAG,1990,0.43 28\r\nGISTEMP,1990,0.44\r\nGCAG,1989,0.297\r\nGISTEMP,1989,0.29\r\nGCAG,1988,0.3757\r \nGISTEMP,1988,0.41\r\nGCAG,1987,0.3696\r\nGISTEMP,1987,0.33\r\nGCAG,1986,0.2296\r\nGI $\verb|STEMP, 1986, 0.19| \verb|r| nGCAG, 1985, 0.1342| \verb|r| nGISTEMP, 1985, 0.12| \verb|r| nGCAG, 1984, 0.149| |r| nGCAG, 0.149| |r| nGCA$ P,1984,0.15\r\nGCAG,1983,0.3411\r\nGISTEMP,1983,0.3\r\nGCAG,1982,0.1815\r\nGISTEMP,198 2,0.13\r\nGCAG,1981,0.2999\r\nGISTEMP,1981,0.33\r\nGCAG,1980,0.2637\r\nGISTEMP,1980,0. 27\r\nGCAG,1979,0.2273\r\nGISTEMP,1979,0.17\r\nGCAG,1978,0.1123\r\nGISTEMP,1978,0.07\r $\label{local_condition} $$ \arrange CAG, 1977, 0.1978 \arrange CAG, 1976, -0.0792 \arrange CAG, 1976, -0.11 \arrange CAG, 1977, 0.1978 \arrange CAG, 1977, 0.1978 \arrange CAG, 1976, -0.0792 \arrange CAG, -0.0792 \a$ GCAG, 1975, 0.0034\r\nGISTEMP, 1975, -0.02\r\nGCAG, 1974, -0.0719\r\nGISTEMP, 1974, -0.07\r\nG CAG, 1973, 0.1641\r\nGISTEMP, 1973, 0.15\r\nGCAG, 1972, 0.0264\r\nGISTEMP, 1972, 0.01\r\nGCAG, 1971,-0.0783\r\nGISTEMP,1971,-0.09\r\nGCAG,1970,0.0372\r\nGISTEMP,1970,0.02\r\nGCAG,19 69,0.0929\r\nGISTEMP,1969,0.07\r\nGCAG,1968,-0.0296\r\nGISTEMP,1968,-0.07\r\nGCAG,196 7,-0.0131\r\nGISTEMP,1967,-0.02\r\nGCAG,1966,-0.0227\r\nGISTEMP,1966,-0.05\r\nGCAG,196 5,-0.078\r\nGISTEMP,1965,-0.1\r\nGCAG,1964,-0.1495\r\nGISTEMP,1964,-0.2\r\nGCAG,1963, 0.1068\r\nGISTEMP,1963,0.06\r\nGCAG,1962,0.0888\r\nGISTEMP,1962,0.03\r\nGCAG,1961,0.07 \r\nGISTEMP,1959,0.03\r\nGCAG,1958,0.1095\r\nGISTEMP,1958,0.07\r\nGCAG,1957,0.0488\r\n GISTEMP, 1957, 0.04\r\nGCAG, 1956, -0.199\r\nGISTEMP, 1956, -0.2\r\nGCAG, 1955, -0.1354\r\nGIS TEMP, 1955, -0.15\r\nGCAG, 1954, -0.1165\r\nGISTEMP, 1954, -0.13\r\nGCAG, 1953, 0.0952\r\nGIST EMP,1953,0.08\r\nGCAG,1952,0.0248\r\nGISTEMP,1952,0.01\r\nGCAG,1951,-0.0132\r\nGISTEM P,1951,-0.07\r\nGCAG,1950,-0.1616\r\nGISTEMP,1950,-0.18\r\nGCAG,1949,-0.0568\r\nGISTEM P,1949,-0.09\r\nGCAG,1948,-0.0487\r\nGISTEMP,1948,-0.09\r\nGCAG,1947,-0.0477\r\nGISTEM P,1947,-0.05\r\nGCAG,1946,-0.004\r\nGISTEMP,1946,-0.04\r\nGCAG,1945,0.171\r\nGISTEMP,1 945,0.12\r\nGCAG,1944,0.2928\r\nGISTEMP,1944,0.25\r\nGCAG,1943,0.157\r\nGISTEMP,1943, 0.13\r\nGCAG,1942,0.1538\r\nGISTEMP,1942,0.09\r\nGCAG,1941,0.196\r\nGISTEMP,1941,0.12 \r\nGCAG,1940,0.0947\r\nGISTEMP,1940,0.08\r\nGCAG,1939,-0.0139\r\nGISTEMP,1939,-0.03\r \nGCAG,1938,-0.0288\r\nGISTEMP,1938,-0.03\r\nGCAG,1937,-0.0157\r\nGISTEMP,1937,-0.03\r \nGCAG,1936,-0.1134\r\nGISTEMP,1936,-0.15\r\nGCAG,1935,-0.1392\r\nGISTEMP,1935,-0.2\r \nGCAG,1934,-0.1015\r\nGISTEMP,1934,-0.14\r\nGCAG,1933,-0.2439\r\nGISTEMP,1933,-0.29\r \ngCAG,1932,-0.1168\r\ngISTEMP,1932,-0.17\r\ngCAG,1931,-0.0686\r\ngISTEMP,1931,-0.09\r \nGCAG,1930,-0.1003\r\nGISTEMP,1930,-0.15\r\nGCAG,1929,-0.2985\r\nGISTEMP,1929,-0.36\r $\nGCAG, 1928, -0.1774\r\nGISTEMP, 1928, -0.21\r\nGCAG, 1927, -0.1546\r\nGISTEMP, 1927, -0.21\r\nGCAG, 1928, -0.1546\r\nGISTEMP, 1927, -0.21\r\nGCAG, 1928, -0.1546\r\nGISTEMP, 1928, -0.21\r\nGCAG, 1928, -0.1546\r\nGISTEMP, 1928, -0.21\r\nGCAG, 1928, -0.1546\r\nGISTEMP, 1927, -0.21\r\nGCAG, 1928, -0.1546\r\nGISTEMP, 1927, -0.21\r\nGCAG, 1928, -0.1546\r\nGISTEMP, 1927, -0.21\r\nGCAG, 1928, -0.1546\r\nGISTEMP, 1927, -0.21\r\nGCAG, 1927, -0.1546\r\nGCAG, 1927, -0.1546\r\nGISTEMP, 1927, -0.21\r\nGCAG, 1927, -0.1546\r\nGCAG, 1927, -0.154$ \nGCAG,1926,-0.0667\r\nGISTEMP,1926,-0.1\r\nGCAG,1925,-0.1481\r\nGISTEMP,1925,-0.21\r \nGCAG,1924,-0.2486\r\nGISTEMP,1924,-0.28\r\nGCAG,1923,-0.2156\r\nGISTEMP,1923,-0.24\r \nGCAG,1922,-0.2304\r\nGISTEMP,1922,-0.28\r\nGCAG,1921,-0.1485\r\nGISTEMP,1921,-0.21\r \nGCAG,1920,-0.2105\r\nGISTEMP,1920,-0.27\r\nGCAG,1919,-0.2055\r\nGISTEMP,1919,-0.22\r \nGCAG,1918,-0.2084\r\nGISTEMP,1918,-0.26\r\nGCAG,1917,-0.3146\r\nGISTEMP,1917,-0.4\r \nGCAG,1916,-0.293\r\nGISTEMP,1916,-0.34\r\nGCAG,1915,-0.0693\r\nGISTEMP,1915,-0.11\r \nGCAG,1914,-0.1395\r\nGISTEMP,1914,-0.16\r\nGCAG,1913,-0.3162\r\nGISTEMP,1913,-0.34\r \nGCAG,1912,-0.3288\r\nGISTEMP,1912,-0.35\r\nGCAG,1911,-0.4332\r\nGISTEMP,1911,-0.44\r $\label{local_condition} $$ \nGCAG, 1910, -0.3789 \r \nGISTEMP, 1910, -0.42 \r \nGCAG, 1909, -0.4261 \r \nGISTEMP, 1909, -0.47 \r \nGCAG, 1910, -0.3789 \r \nGISTEMP, 1910, -0.42 \r \nGCAG, 1909, -0.4261 \r \nGISTEMP, 1909, -0.47 \r \nGISTEMP, 1909, -0$ \nGCAG,1908,-0.4396\r\nGISTEMP,1908,-0.43\r\nGCAG,1907,-0.3706\r\nGISTEMP,1907,-0.4\r \nGCAG,1906,-0.2174\r\nGISTEMP,1906,-0.23\r\nGCAG,1905,-0.2931\r\nGISTEMP,1905,-0.28\r \nGCAG,1904,-0.4194\r\nGISTEMP,1904,-0.44\r\nGCAG,1903,-0.3369\r\nGISTEMP,1903,-0.35\r \nGCAG,1902,-0.2463\r\nGISTEMP,1902,-0.27\r\nGCAG,1901,-0.1417\r\nGISTEMP,1901,-0.15\r \ngCAG,1900,-0.0679\r\nGISTEMP,1900,-0.09\r\nGCAG,1899,-0.1173\r\nGISTEMP,1899,-0.16\r \ngCAG,1898,-0.2546\r\ngISTEMP,1898,\r\ngCAG,1897,\r\ngISTEMP,1897,\r\ngCAG,1896,-0.09 $74 \\ r \\ nGISTEMP, 1896, -0.15 \\ r \\ nGCAG, 1895, -0.229 \\ r \\ nGISTEMP, 1895, -0.21 \\ r \\ nGCAG, 1894, -0.280 \\ r \\ nGISTEMP, 1895, -0.21 \\ r \\ nGCAG, 1894, -0.280 \\ r \\ nGISTEMP, 1895, -0.21 \\ r \\ nGCAG, 1894, -0.280 \\ r \\ nGISTEMP, 1895, -0.21 \\ r \\ nGCAG, 1894, -0.280 \\ r \\ nGISTEMP, 1895, -0.21 \\ r \\ nGCAG, 1894, -0.280 \\ r \\ nGISTEMP, 1895, -0.21 \\ r \\ nGCAG, 1894, -0.280 \\ r \\ nGISTEMP, 1895, -0.21 \\ r \\ nGCAG, 1894, -0.280 \\ r \\ nGISTEMP, 1895, -0.21 \\ r \\ nGCAG, 1894, -0.280 \\ r \\ nGCAG, -0.280 \\ r \\$ 8\r\nGISTEMP,1894,-0.31\r\nGCAG,1893,-0.3212\r\nGISTEMP,1893,-0.3\r\nGCAG,1892,-0.3062 \r\nGISTEMP,1892,-0.27\r\nGCAG,1891,-0.2532\r\nGISTEMP,1891,-0.24\r\nGCAG,1890,-0.322 \r\nGISTEMP,1890,-0.37\r\nGCAG,1889,-0.0982\r\nGISTEMP,1889,-0.12\r\nGCAG,1888,-0.1471 \r\nGISTEMP,1888,-0.2\r\nGCAG,1887,-0.2489\r\nGISTEMP,1887,-0.33\r\nGCAG,1886,-0.2003 \r\nGISTEMP,1886,-0.31\r\nGCAG,1885,-0.2125\r\nGISTEMP,1885,-0.32\r\nGCAG,1884,-0.2009 \r\nGISTEMP,1884,-0.28\r\nGCAG,1883,-0.1424\r\nGISTEMP,1883,-0.21\r\nGCAG,1882,-0.0648 \r\nGISTEMP,1882,-0.1\r\nGCAG,1881,-0.0628\r\nGISTEMP,1881,-0.12\r\nGCAG,1880,-0.1148 \r\nGISTEMP, 1880, -0.2\r\n'} Remove unwanted files... %ls annual temp2.csv 'annual temp csv2 (2).csv' sample data/ 'annual temp csv2 (1).csv' annual temp csv2.csv %rm *.csv %pycat annual_temp_csv2.csv Make a DataFrame object to hold the contents of the data set DataFrame help page Just like with the pd.Series call, you can specify the data, index labels (row labels in this case) In addition to row labels, you can also specify column labels (with 'columns') Can also specify data type (default is inferred) • If you read in the data from a csv file, you will be able to inheret row and column labels (if they are specified in the file). # make the call to pd.DataFrames to create the DF - usage much like pd.Series df = pd.read csv('annual temp csv2.csv') # take a look at the output... # compare to print(df) - looks nicer with display thanks to iPython backend display(df) Mean Source Year GCAG 2016 0.9363 **1** GISTEMP 2016 NaN GCAG 2015 2 NaN **3** GISTEMP 2015 0.8700 4 GCAG 2014 0.7408 269 GISTEMP 1882 -0.1000 GCAG 1881 -0.0628 270 **271** GISTEMP 1881 -0.1200 GCAG 1880 -0.1148 272 273 GISTEMP 1880 -0.2000 274 rows × 3 columns # another handy display function...good for large dfs that are too big to fit -# at least you can get an idea of the overall structure df.head() Source Year Mean GCAG 2016 0.9363 **1** GISTEMP 2016 NaN GCAG 2015 NaN GISTEMP 2015 0.8700 GCAG 2014 0.7408 Get a high-level summary of the data using built-in functionality of DataFrame object API reference page What do you notice about the two counts for Year and for Mean df.describe() Year Mean 274.000000 269.000000 count mean 1948.000000 0.032148 39.619805 0.312434 std 1880.000000 -0.470000 min 25% 1914.000000 -0.205500 **50%** 1948.000000 -0.050000 **75%** 1982.000000 0.227300 **max** 2016.000000 0.936300 Just like with Series object, can compute mean, std, etc df['Year'].mean() Out[]: 1948.0 remember that you can also call by field...I prefer by name like ['Mean'] to avoid confusion with built in methods/functions, but either will work #df['Mean'].std() #df.Mean.std() df['Mean'].mean() Out[]: 0.032147583643122576 By default, mean, std etc will skip (ignore) missing values (NaNs) Sometimes, its good to do a sanity check if you think there are missing values. • Can do this by chosing to NOT skip the NaNs...in which case if they exist you'll get back NaN as the answer! Then you know that there are NaNs in the data set. df['Mean'].mean(skipna=False) Out[]: nan Find missing values in your data and deal with them (NaNs) · Can apply to just one column at a time note that you can call the isna method from the object directly To make this work, you index into the data frame where 'Mean' is a Nan # isolate just the rows (indicies) where Mean is NaN df['Mean'].isna() Out[]: 0 False True True 3 False False False 269 270 False 271 False 272 False False 273 Name: Mean, Length: 274, dtype: bool # now index into df using the true/false sequence from above! df[df['Mean'].isna()] Source Year Mean **1** GISTEMP 2016 NaN 2 GCAG 2015 NaN **237** GISTEMP 1898 NaN GCAG 1897 238 NaN **239** GISTEMP 1897 NaN Or do the opposite, isolate just the rows where Mean is not NaN (i.e. its a real number) df[df['Mean'].notna()] Out[]: Source Year Mean GCAG 2016 0.9363 **GISTEMP** 2015 0.8700 3 2014 0.7408 GCAG GISTEMP 2014 0.7400 GCAG 2013 6 0.6679 GISTEMP 1882 **GCAG** -0.0628 270 1881 **271** GISTEMP 1881 -0.1200GCAG 1880 272 -0.1148273 GISTEMP 1880 -0.2000 269 rows × 3 columns Can deal with NaNs lots of ways... · Can make a new DF without them can assign the mean of all of the data to NaNs # make a new df, but only keep the non-NaN entries df2 = df[df['Mean'].notna()] df2.head() Source Year Mean GCAG 2016 0.9363 0 3 GISTEMP 2015 0.8700 GCAG 2014 0.7408 GISTEMP 2014 0.7400 5 GCAG 2013 0.6679 Fill the NaN with the mean of the column! Or any other value...just pass it into fillna • see also 'interpolate' for more functions like this NEED TO ASSIGN output to apply changes..e.g df = df.fillna(....) print(df['Mean'].mean()) df2=df.fillna(df['Mean'].mean()) df2.head() 0.032147583643122576 Source Year Mean GCAG 2016 0.936300 **1** GISTEMP 2016 0.032148 GCAG 2015 0.032148 GISTEMP 2015 0.870000 GCAG 2014 0.740800 Pull out selected data and remove from DF df.head() Source Year Mean GCAG 2016 0.9363 1 GISTEMP 2016 NaN 2 GCAG 2015 NaN GISTEMP 2015 0.8700 GCAG 2014 0.7408 # or your could self assign df = df[] here to update existing data frame df2 = df[df['Source'] == 'GISTEMP'] df2.head() Source Year Mean **1** GISTEMP 2016 NaN **3** GISTEMP 2015 0.87 **5** GISTEMP 2014 0.74 **7** GISTEMP 2013 0.65 **9** GISTEMP 2012 0.63 Grab a range of rows...across a set of years, for example df2 = df[(df['Year']>1990) & (df['Year']<=2015)]</pre> display(df2) Source Year Mean GCAG 2015 NaN GISTEMP 2015 0.8700 GCAG 2014 0.7408 **GISTEMP** 2014 0.7400 GCAG 2013 0.6679 GISTEMP 2013 0.6500 2012 0.6240 GCAG GISTEMP 2012 0.6300 10 GCAG 2011 0.5788 GISTEMP 2011 0.6000 0.7014 12 GCAG 2010 GISTEMP 2010 0.7100 GCAG 2009 0.6367 GISTEMP 2009 0.6400 GCAG 2008 0.5419 GISTEMP 2008 0.5400 17 GCAG 2007 0.6100 GISTEMP 2007 0.6600 20 GCAG 2006 0.6125 GISTEMP 2006 0.6300 GCAG 2005 0.6585 GISTEMP 2005 0.6900 24 GCAG 2004 0.5783 GISTEMP 2004 0.5500 25 26 GCAG 2003 0.6134 27 GISTEMP 2003 0.6200 GCAG 2002 0.6023 28 GISTEMP 2002 0.6300 29 GCAG 2001 0.5473 31 GISTEMP 2001 0.5500 GCAG 2000 0.4262 32 **GISTEMP** 2000 0.4200 33 GCAG 1999 0.4438 34 35 GISTEMP 1999 0.4200 36 GCAG 1998 0.6344 **GISTEMP** 1998 0.6400 37 **GCAG** 1997 0.5187 39 GISTEMP 1997 0.4800 1996 0.3228 40 GCAG **GISTEMP** 1996 0.3500 GCAG 1995 0.4577 43 GISTEMP 1995 0.4600 44 1994 0.3409 GCAG **GISTEMP** 1994 0.3200 45 GCAG 1993 0.2853 47 GISTEMP 1993 0.2400 1992 48 GCAG 0.2571 49 **GISTEMP** 1992 0.2300 GCAG 1991 0.4055 GISTEMP 1991 0.4300 Apply several filters at once! Be careful here - readability of code is the prime directive...don't write one-liners that are so dense that nobody can understand them! df2 = df[(df['Source']=='GCAG') & (df['Year']>1990) & (df['Year']<=2015)]display(df2) More on indexing and selection of specific coordinates in a DF Row selection - this is a bit more complex as there are many methods You can use df.loc to select a row by its label name You can use df.iloc to select a row by its integer location (from 0 to length-1 of the axis) You can use boolean vectors to select a set of rows that satisfy some condition Contrary to usual slicing conventions, both the start and the stop indices are included when using the DF.LOC option...see below for demo. This makes sense because you're indexing by label name, not by a zero-based integer index. #for the next steps, load annual temp2 files.upload() Upload widget is only available when the cell has been executed in Choose Files No file chosen the current browser session. Please rerun this cell to enable. Saving annual temp2.csv to annual temp2 (1).csv {'annual temp2.csv': b'Year, Mean, Std\r\n2016, 0.9363, 0.751983201423804\r\n2015,,\r\n201 4,0.7408,0.308372327229529\r\n2013,0.6679,0.0642260963534021\r\n2012,0.624,0.369924308 791563\r\n2011,0.5788,0.519886611755557\r\n2010,0.7014,0.00554037371614102\r\n2009,0.6 367,0.710221086775733\r\n2008,0.5419,0.13596775970467\r\n2007,0.61,0.600989623657847\r \n2006,0.6125,0.477444551272558\r\n2005,0.6585,0.200089544839782\r\n2004,0.5783,0.3506 $36861158391\r\n2003, 0.6134, 0.175530439616115\r\n2002, 0.6023, 0.958127276637697\r\n2001, 0.6023, 0.958127276637697\r\n2001, 0.958127697697\r\n2001, 0.9581277697697\r\n2001, 0.9581277697697\r\n2001, 0.958127697\r\n2001, 0.958127697\r\n2001, 0.958127697\r\n2001, 0.958127697\r\n2001, 0.958127697\r\n2001, 0.958127697\r\n2001, 0.958127697\r\n2001, 0.958127697\r\n2001, 0.9581277697\r\n2001, 0.958127697\r\n2001, 0.958127697\r$ 891\r\n1998,0.6344,0.318651207090184\r\n1997,0.5187,0.793294701283796\r\n1996,0.3228, \n1993,0.2853,0.882884329517337\r\n1992,0.2571,0.562569395616625\r\n1991,0.4055,0.5884 $29206506485 \\ \text{r} \\ \text{n} \\ 1990, \\ 0.4328, \\ 0.147454693380858 \\ \text{r} \\ \text{n} \\ 1989, \\ 0.297, \\ 0.721697423072878 \\ \text{r} \\ \text{n} \\ 1988, \\ 0.297, \\ 0.721697423072878 \\ \text{r} \\ \text{n} \\ 1988, \\ 0.297, \\ 0.721697423072878 \\ \text{r} \\ \text{n} \\ 1988, \\ 0.297, \\ 0.721697423072878 \\ \text{r} \\ \text{n} \\ 1988, \\ 0.297, \\ 0.721697423072878 \\ \text{r} \\ \text{n} \\ 1988, \\ 0.297, \\ 0.721697423072878 \\ \text{r} \\ \text{n} \\ 1988, \\ 0.297, \\ 0.2$ $0.3757, 0.211935846229154 \\ r\\ n1987, 0.3696, 0.645365247533375 \\ r\\ n1986, 0.2296, 0.357759300785 \\ r\\ n1987, 0.3696, 0.645365247533375 \\ r\\ n1988, 0.2296, 0.357759300785 \\ r\\ n1987, 0.3696, 0.645365247533375 \\ r\\ n1988, 0.2296, 0.357759300785 \\ r\\ n1988, 0.2296, 0.357759300785 \\ r\\ n1989, 0.3696, 0.645365247533375 \\ r\\ n1988, 0.2296, 0.357759300785 \\ r\\ n1989, 0.3696,$ $8325 \\ r \\ n1985, 0.1342, 0.344254799672039 \\ r \\ n1984, 0.149, 0.772079944607475 \\ r \\ n1983, 0.34111, 0.341111, 0.341111, 0.341111, 0.341111, 0.341111, 0.341111, 0.3411111, 0.3411111, 0.34111111, 0.3411111, 0.3411111, 0.341111111111111$ $0.0687893670043981 \\ r\\ n1982, 0.1815, 0.511231020671473 \\ r\\ n1981, 0.2999, 0.0188473801261881$ \r\n1980,0.2637,0.201064607422642\r\n1979,0.2273,0.180923535127335\r\n1978,0.1123,0.36 0679532662809\r\n1977,0.1978,0.688585411771921\r\n1976,-0.0792,0.200876803208843\r\n19 $8734193673 \\ r\\ n1972, 0.0264, 0.956011291618283 \\ r\\ n1971, -0.0783, 0.864109153001867 \\ r\\ n1970, 0.0783, 0.$ 77688\r\n1967,-0.0131,0.243417247260451\r\n1966,-0.0227,0.392410718324416\r\n1965,-0.0 78,0.231828574650785\r\n1964,-0.1495,0.140240270498326\r\n1963,0.1068,0.24013745909740 8\r\n1962,0.0888,0.529270041060965\r\n1961,0.0775,0.519028686374525\r\n1960,0.0204,0.1 56221801697547\r\n1959,0.0596,0.726664204051616\r\n1958,0.1095,0.162915134056691\r\n19 57,0.0488,0.999169935961326\r\n1956,-0.199000000000000,0.546784392995795\r\n1955,-0. 1354,0.171323144951672\r\n1954,-0.1165,0.48231483549844\r\n1953,0.0952,0.1417283380719 8\r\n1952,0.0248,0.673033865260092\r\n1951,-0.0132,0.650865462586769\r\n1950,-0.1616, \n1947,-0.0477,0.820574222667296\r\n1946,-0.004,0.148684153559589\r\n1945,0.171,0.0463 47412074689\r\n1944,0.2928,0.16891121594668\r\n1943,0.157,0.566967109683963\r\n1942,0. $1538, 0.519371976292688 \\ r \\ n1941, 0.196, 0.851964242289424 \\ r \\ n1940, 0.0947, 0.007576144706041 \\ r \\ n1940, 0.0947, 0.00757614 \\ r \\ n1940, 0.0076, 0.0076761 \\ r \\ n1940, 0.0076$ 7,0.291027811409836\r\n1936,-0.1134,0.503254927274545\r\n1935,-0.1392,0.00460569729445 638\r\n1934,-0.1015,0.604094168246076\r\n1933,-0.2439,0.981605361042585\r\n1932,-0.116 9\r\n1929,-0.2985,0.909757573806934\r\n1928,-0.1774,0.692378986542129\r\n1927,-0.1546, \r\n1924,-0.2486,0.214365228374656\r\n1923,-0.2156,0.991481274558267\r\n1922,-0.2304, \r\n1919,-0.2055,0.0804197154682573\r\n1918,-0.2084,0.214025254716516\r\n1917,-0.3146, 0.342808847360293\r\n1916,-0.293,0.522969158250288\r\n1915,-0.0693,0.273753723961632\r \n1914,-0.1395,0.815101456578759\r\n1913,-0.3162,0.165023232641638\r\n1912,-0.3288,0.3 10420139528037\r\n1911,-0.4332,0.693082026953693\r\n1910,-0.3789,0.865678454926031\r\n $1909, -0.4261, 0.623919733722176 \verb|\r| n1908, -0.4396, 0.438704923193233 \verb|\r| n1907, -0.3706, 0.27386, 0.27386, 0.43870492319323 \verb|\r| n1907, -0.3706, 0.27386, 0.4387049231932 \verb|\r| n1907, -0.3706, 0.43870492319323 \verb|\r| n1907, -0.3706, 0.4387049231932 \verb|\r| n1907, -0.3706, 0.438704923 \verb|\r| n1907, -0.3706, 0.43870492 \verb|\r| n1907, -0.3706, 0.43870492 \verb|\r| n1907, -0.3706, 0.43870492 \verb|\r| n1907, -0.3706, 0.43870492 |\r| n1907, -0.3706, 0.43870492 |\r| n1907, -0.3706, 0.4387049$ 439136137311\r\n1906,-0.2174,0.0364778482638509\r\n1905,-0.2931,0.424024934062817\r\n1 904,-0.4194,0.979370498983175\r\n1903,-0.3369,0.279523116904304\r\n1902,-0.2463,0.0077 3229964501328\r\n1901,-0.1417,0.0457725939584814\r\n1900,-0.0679,0.0678002958632341\r \n1899,-0.1173,0.19280587714598\r\n1898,-0.2546,0.481234008629767\r\n1897,,0.835940846 940579\r\n1896,-0.0974,0.488648240249878\r\n1895,-0.229000000000000,0.52592789053282 2\r\n1894,-0.2808,0.70011900069477\r\n1893,-0.3212,0.399573260715036\r\n1892,-0.3062, \r\n1889,-0.0982,0.953727894734898\r\n1888,-0.1471,0.376836477660656\r\n1887,-0.2489, $0.309235336659661 \\ r \\ 11886, -0.2003, 0.555201311487298 \\ r \\ 11885, -0.2125, 0.361183756912452$ \r\n1884,-0.2009,0.882296486501537\r\n1883,-0.1424,0.146379303884453\r\n1882,-0.0648, 0.513377910557089\r\n1881,-0.0628,0.180162696314947\r\n1880,-0.1148,0.82395270652697 **5'**} %pycat annual temp2.csv df = pd.read csv('annual temp2.csv', index col=0) In [4]: df.head() Std Out[4]: Mean Year 0.9363 2016 0.751983 2015 NaN NaN **2014** 0.7408 0.308372 **2013** 0.6679 0.064226 2012 0.6240 0.369924 This returns the data associated with one row df.loc[2014] 0.740800 Out[]: Mean Std 0.308372 Name: 2014, dtype: float64 this returns the rows associated with a set of years specified in a list # non-contiguous entries df2 = df.loc[[1999, 2015, 2016, 1880]]df2.head() Mean Std Year **1999** 0.4438 0.583837 2015 NaN NaN **2016** 0.9363 0.751983 **1880** -0.1148 0.823953 df.head() Mean Std Year **2016** 0.9363 0.751983 **2015** NaN NaN **2014** 0.7408 0.308372 **2013** 0.6679 0.064226 **2012** 0.6240 0.369924 # note that years run in descending order... df.loc[2014:2016] Mean Std Year # but this will work... df.loc[2016:2010:2] Mean Std Year **2016** 0.9363 0.751983 **2014** 0.7408 0.308372 2012 0.6240 0.369924 **2010** 0.7014 0.005540 # flip the data frame upside down df=df.loc[::-1]df.head() Mean Std Year **1880** -0.1148 0.823953 **1881** -0.0628 0.180163 **1882** -0.0648 0.513378 **1883** -0.1424 0.146379 **1884** -0.2009 0.882296 # now the years run in order df.loc[1990:2016] Std Mean Year **1990** 0.4328 0.147455 **1991** 0.4055 0.588429 1992 0.2571 0.562569 1993 0.2853 0.882884 **1994** 0.3409 0.484950 **1995** 0.4577 0.179344 **1996** 0.3228 0.297116 1997 0.5187 0.793295 **1998** 0.6344 0.318651 1999 0.4438 0.583837 **2000** 0.4262 0.315162 **2001** 0.5473 0.654812 **2002** 0.6023 0.958127 2003 0.6134 0.175530 2004 0.5783 0.350637 **2005** 0.6585 0.200090 **2006** 0.6125 0.477445 **2007** 0.6100 0.600990 2008 0.5419 0.135968 2009 0.6367 0.710221 **2010** 0.7014 0.005540 **2011** 0.5788 0.519887 **2012** 0.6240 0.369924 **2013** 0.6679 0.064226 2014 0.7408 0.308372 2015 NaN NaN **2016** 0.9363 0.751983 # select rows and select colums #df.loc[2016:2014, 'Mean':'Std'] df.loc[2014:2016, 'Mean':'Std'] Out[]: Year 2014 0.7408 2015 NaN 0.9363 2016 Name: Mean, dtype: float64 iloc does indexing by row location (not label) use normal rules of slicing here...start:stop:step In [8]: # the first 10 df.iloc[:10:2] # # reverse # df.iloc[::-1] # # every other # df.iloc[::2] Out[8]: 0.751983201423804 Adding a column is easy and can be done dynamically (on the fly) Make a new column of True and False to mark years above/below mean temp deviation mean temp = df['Mean'].mean() print('mean temp:', mean temp) # then populate the new column df['HighLow'] = df['Mean']>mean temp df.head() mean temp: 0.043760740740740725 Std HighLow Mean Year **1880** -0.1148 0.823953 False **1881** -0.0628 0.180163 False **1882** -0.0648 0.513378 False **1883** -0.1424 0.146379 False **1884** -0.2009 0.882296 False If you want to convert values in a column, can be a little tricky... Use what you might think is the intuitive way to convert True to 1 and False to 0 in our new column 'HighLow' This throws a weird warning because you're trying to modify the thing that you're using as an df.HighLow[df.HighLow==True] = 1 #df['HighLow'] df.HighLow[df.HighLow==False] = 0 df.head() /usr/local/lib/python3.7/dist-packages/ipykernel launcher.py:1: SettingWithCopyWarnin A value is trying to be set on a copy of a slice from a DataFrame See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/use r guide/indexing.html#returning-a-view-versus-a-copy """Entry point for launching an IPython kernel. Std HighLow Year **2016** 0.9363 0.751983 1.0 NaN 2015 NaN 0.0 **2014** 0.7408 0.308372 1.0 **2013** 0.6679 0.064226 **2012** 0.6240 0.369924 1.0 Solution - use .loc to return the information and then modify it # reload our df, re-create a new version of our HighLow column df = pd.read csv('annual temp2.csv', index col=0) # make our new column again. mean temp = df['Mean'].mean() print('mean temp:', mean temp) # then populate the new column df['HighLow'] = df['Mean']>mean temp df.head() mean temp: 0.04376074074074073 Std HighLow Year **2016** 0.9363 0.751983 True 2015 NaN NaN False **2014** 0.7408 0.308372 True **2013** 0.6679 0.064226 True **2012** 0.6240 0.369924 True like this...using the .loc method ensures that things don't get confused... df.loc[df['HighLow'] == True, 'HighLow'] = 1 df.loc[df['HighLow'] == False, 'HighLow'] = 0 df.head() Std HighLow Mean Year **2016** 0.9363 0.751983 1 2015 NaN NaN **2014** 0.7408 0.308372 1 **2013** 0.6679 0.064226 1 2012 0.6240 0.369924 1 Access specific row and columns! # flip it over to chronological order so its a little more intuitive to slice df = df.loc[::-1]df.head() Mean Std HighLow Year

Pandas DataFrames

1880 -0.1148 0.823953 False **1881** -0.0628 0.180163 False **1882** -0.0648 0.513378 False **1883** -0.1424 0.146379 False **1884** -0.2009 0.882296 False In []: # just data from one year from just the "Mean" column df.loc[2016, 'Mean'] Out[]: 0.9363 In []: # a range of years df.loc[1880:1884, 'Mean'] Out[]: Year 1880 -0.1148 1881 -0.0628 1882 -0.0648 1883 -0.1424 1884 -0.2009 Name: Mean, dtype: float64 In []: # range of rows and columns df.loc[1880:1884, ['Mean','HighLow']] Mean HighLow Out[]: Year False **1880** -0.1148 **1881** -0.0628 False **1882** -0.0648 False **1883** -0.1424 False **1884** -0.2009 False In []: # note that the order in which you ask for columns impacts the output $row_ind = [1880, 1980]$ col_ind = ['HighLow', 'Mean'] df2 = df.loc[row_ind, col_ind] df2.head() Out[]: HighLow Mean Year **1880** False -0.1148 1980 True 0.2637 A few more examples of using iloc to index into specific rows, columns In [54]: df.head() Mean Std Out[54]: Year **2016** 0.9363 0.751983 **2015** NaN NaN **2014** 0.7408 0.308372 **2013** 0.6679 0.064226 **2012** 0.6240 0.369924 In [58]: df.iloc[:20:2] Std Out[58]: Mean Year **2016** 0.9363 0.751983 **2014** 0.7408 0.308372 **2012** 0.6240 0.369924 **2010** 0.7014 0.005540 2008 0.5419 0.135968 **2006** 0.6125 0.477445 **2004** 0.5783 0.350637 **2002** 0.6023 0.958127 **2000** 0.4262 0.315162 **1998** 0.6344 0.318651 In [63]: df.iloc[:10, 0] Out[63]: Year 2016 0.9363 2015 NaN 2015 NaN 2014 0.7408 2013 0.6679 2012 0.6240 2011 0.5788 2010 0.7014 2009 0.6367 2008 0.5419 2007 0.6100 Name: Mean, dtype: float64 Deleting columns... In [68]: df_temp = df.copy() In [69]: df_temp.head() Mean Std Year **2016** 0.9363 0.751983 NaN **2015** NaN **2014** 0.7408 0.308372 **2013** 0.6679 0.064226 **2012** 0.6240 0.369924 In [70]: # using the del command will delete a column from the DF # make an explicit copy! (why???) so we don't overwrite our original data frame $df2 = df_{temp.copy()}$ del df2['Mean'] df2.head() Std Year 2016 0.751983 2015 NaN **2014** 0.308372 **2013** 0.064226 2012 0.369924 df temp.head() Mean Std Year **2016** 0.9363 0.751983 2015 NaN NaN **2014** 0.7408 0.308372 **2013** 0.6679 0.064226 **2012** 0.6240 0.369924 Replace method • first param is what you want to replace (can be any data type as needed) • second param is what you want to replace it with In [72]: # make a df from a dictionary data = $\{'D1' : [6,8,4,3], 'D2' : [4,4,2,1]\}$ df = pd.DataFrame(data) df.head() D1 D2 0 6 4 2 4 3 1 In [75]: df = df.replace(4, 7) df.head() D1 D2 7 **1** 8 2 7 2 3 1 Set index from a column in a data frame In [79]: data = {'D1': [6,8,4,3], 'D2': [4,4,2,1], 'D3': [2018,2019,2020,2021]} df = pd.DataFrame(data) df.head() D1 D2 D3 **2018** 6 4 2018 **2019** 8 4 2019 **2020** 4 2 2020 **2021** 3 1 2021 In [78]: df = df.set_index('D3') df.head() Out[78]: D1 D2 D3 **2018** 6 4 **2019** 8 4 **2020** 4 2 **2021** 3 1 Loop over data frame and compute the mean of 'w' consecutive rows In [86]: # make a df import random as random random.seed(10) $n_{data_pnts} = 100$ d1**=**[] d2**=**[] for i in range(n_data_pnts): d1.append(random.randint(0,40)) d2.append(random.randint(0,30)) # then make a data frame df = pd.DataFrame({'D1':d1, 'D2':d2}) df.head() Out[86]: **D1 D2 0** 36 1 **1** 27 15 **2** 36 0 **3** 13 14 **4** 31 26 In [100... win = 20 # moving average window n = len(df)# init a list to append moving average m_avg = [] # init a counter to keep track of where we are in the DF for i in range(0,n,win): m_avg.append(df.loc[i:i+win,'D1'].mean()) # print out our list of windowed averages print(m_avg) print(len(m_avg)) [21.714285714285715, 17.952380952380953, 17.476190476190474, 22.61904761904762, 21.1] Preview of next week - quick intro to plotting In [94]: import matplotlib.pyplot as plt plt.plot(df['D1']) plt.show() plt.plot(m_avg) plt.show() 40 35 30 25 15 10 5 0 60 ò 20 40 80 100 22 21 20 19 18 3.5 0.5 2.5 1.0 1.5 2.0 3.0 4.0 0.0