BasicStatisticalTesting

June 7, 2022

In this lecture we're going to review some of the basics of statistical testing in python. We're going to talk about hypothesis testing, statistical significance, and using scipy to run student's t-tests.

[1]:
```
# We use statistics in a lot of different ways in data science, and on this␣
↪lecture, I want to refresh your
# knowledge of hypothesis testing, which is a core data analysis activity␣
↪behind experimentation. The goal of
# hypothesis testing is to determine if, for instance, the two different␣
↪conditions we have in an experiment
# have resulted in different impacts

# Let's import our usual numpy and pandas libraries
import numpy as np
import pandas as pd

# Now let's bring in some new libraries from scipy
from scipy import stats
```

[2]:
```
# Now, scipy is an interesting collection of libraries for data science and␣
↪you'll use most or perpahs all of
# these libraries. It includes numpy and pandas, but also plotting libraries␣
↪such as matplotlib, and a
# number of scientific library functions as well
```

[48]:
```
# When we do hypothesis testing, we actually have two statements of interest:␣
↪the first is our actual
# explanation, which we call the alternative hypothesis, and the second is that␣
↪the explanation we have is not
# sufficient, and we call this the null hypothesis. Our actual testing method␣
↪is to determine whether the null
# hypothesis is true or not. If we find that there is a difference between␣
↪groups, then we can reject the null
# hypothesis and we accept our alternative.

# Let's see an example of this; we're going to use some grade data
df=pd.read_csv ('datasets/grades.csv')
df.head()
```

[48]:

```
                             student_id  assignment1_grade  \
0  B73F2C11-70F0-E37D-8B10-1D20AFED50B1          92.733946
1  98A0FAE0-A19A-13D2-4BB5-CFBFD94031D1          86.790821
2  D0F62040-CEB0-904C-F563-2F8620916C4E          85.512541
3  FFDF2B2C-F514-EF7F-6538-A6A53518E9DC          86.030665
4  5ECBEEB6-F1CE-80AE-3164-E45E99473FB4          64.813800

          assignment1_submission  assignment2_grade  \
0  2015-11-02 06:55:34.282000000          83.030552
1  2015-11-29 14:57:44.429000000          86.290821
2  2016-01-09 05:36:02.389000000          85.512541
3  2016-04-30 06:50:39.801000000          68.824532
4  2015-12-13 17:06:10.750000000          51.491040

          assignment2_submission  assignment3_grade  \
0  2015-11-09 02:22:58.938000000          67.164441
1  2015-12-06 17:41:18.449000000          69.772657
2  2016-01-09 06:39:44.416000000          68.410033
3  2016-04-30 17:20:38.727000000          61.942079
4  2015-12-14 12:25:12.056000000          41.932832

          assignment3_submission  assignment4_grade  \
0  2015-11-12 08:58:33.998000000          53.011553
1  2015-12-10 08:54:55.904000000          55.098125
2  2016-01-15 20:22:45.882000000          54.728026
3  2016-05-12 07:47:16.326000000          49.553663
4  2015-12-29 14:25:22.594000000          36.929549

          assignment4_submission  assignment5_grade  \
0  2015-11-16 01:21:24.663000000          47.710398
1  2015-12-13 17:32:30.941000000          49.588313
2  2016-01-11 12:41:50.749000000          49.255224
3  2016-05-07 16:09:20.485000000          49.553663
4  2015-12-28 01:29:55.901000000          33.236594

          assignment5_submission  assignment6_grade  \
0  2015-11-20 13:24:59.692000000          38.168318
1  2015-12-19 23:26:39.285000000          44.629482
2  2016-01-11 17:31:12.489000000          44.329701
3  2016-05-24 12:51:18.016000000          44.598297
4  2015-12-29 14:46:06.628000000          33.236594

          assignment6_submission
0  2015-11-22 18:31:15.934000000
1  2015-12-21 17:07:24.275000000
2  2016-01-17 16:24:42.765000000
3  2016-05-26 08:09:12.058000000
```

```
4  2016-01-05 01:06:59.546000000
```

[4]:
```python
# If we take a look at the data frame inside, we see we have six different␣
↪assignments. Lets look at some
# summary statistics for this DataFrame

# Con .format()
print("There are {} rows and {} columns".format(df.shape[0], df.shape[1]))

# Con f'strings
print(f"There are {df.shape[0]} rows and {df.shape[1]} columns")
```

```
There are 2315 rows and 13 columns
There are 2315 rows and 13 columns
```

[5]:
```python
# Vamos a explorar fechas del assignment 1:
df_fechas =  df['assignment1_submission']

# Convierto las fechas a to_datetime()
df_fechas_to_datetime = pd.to_datetime(df_fechas)

print(f'Sin datetime() Max= {df_fechas.max()} Min= {df_fechas.min()}')
print(f'Con datetime() Max= {df_fechas_to_datetime.max()} Min=␣
↪{df_fechas_to_datetime.min()}')

# Se puede ver que Pandas ya detectó las fechas sin utilizar to_datetime()
```

```
Sin datetime() Max= 2016-08-07 18:57:16.825000000 Min= 2015-09-14
23:46:23.696000000
Con datetime() Max= 2016-08-07 18:57:16.825000 Min= 2015-09-14 23:46:23.696000
```

[7]:
```python
# For the purpose of this lecture, let's segment this population into two␣
↪pieces. Let's say those who finish
# the first assignment by the end of December 2015, we'll call them early␣
↪finishers, and those who finish it
# sometime after that, we'll call them late finishers.

early_finishers=df[df['assignment1_submission'] < '2016']

print(len(early_finishers))

early_finishers.head()
```

```
1259
```

[7]:

```
                             student_id  assignment1_grade  \
0  B73F2C11-70F0-E37D-8B10-1D20AFED50B1          92.733946
1  98A0FAE0-A19A-13D2-4BB5-CFBFD94031D1          86.790821
4  5ECBEEB6-F1CE-80AE-3164-E45E99473FB4          64.813800
5  D09000A0-827B-C0FF-3433-BF8FF286E15B          71.647278
8  C9D51293-BD58-F113-4167-A7C0BAFCB6E5          66.595568

          assignment1_submission  assignment2_grade  \
0  2015-11-02 06:55:34.282000000          83.030552
1  2015-11-29 14:57:44.429000000          86.290821
4  2015-12-13 17:06:10.750000000          51.491040
5  2015-12-28 04:35:32.836000000          64.052550
8  2015-12-25 02:29:28.415000000          52.916454

          assignment2_submission  assignment3_grade  \
0  2015-11-09 02:22:58.938000000          67.164441
1  2015-12-06 17:41:18.449000000          69.772657
4  2015-12-14 12:25:12.056000000          41.932832
5  2016-01-03 21:05:38.392000000          64.752550
8  2015-12-31 01:42:30.046000000          48.344809

          assignment3_submission  assignment4_grade  \
0  2015-11-12 08:58:33.998000000          53.011553
1  2015-12-10 08:54:55.904000000          55.098125
4  2015-12-29 14:25:22.594000000          36.929549
5  2016-01-07 08:55:43.692000000          57.467295
8  2016-01-05 23:34:02.180000000          47.444809

          assignment4_submission  assignment5_grade  \
0  2015-11-16 01:21:24.663000000          47.710398
1  2015-12-13 17:32:30.941000000          49.588313
4  2015-12-28 01:29:55.901000000          33.236594
5  2016-01-11 00:45:28.706000000          57.467295
8  2016-01-02 07:48:42.517000000          37.955847

          assignment5_submission  assignment6_grade  \
0  2015-11-20 13:24:59.692000000          38.168318
1  2015-12-19 23:26:39.285000000          44.629482
4  2015-12-29 14:46:06.628000000          33.236594
5  2016-01-11 00:54:13.579000000          57.467295
8  2016-01-03 21:27:04.266000000          37.955847

          assignment6_submission
0  2015-11-22 18:31:15.934000000
1  2015-12-21 17:07:24.275000000
4  2016-01-05 01:06:59.546000000
5  2016-01-20 19:54:46.166000000
```

```
8  2016-01-19 15:24:31.060000000
```

[8]:
```
# So, you have lots of skills now with pandas, how would you go about getting␣
↪the late_finishers dataframe?
# Why don't you pause the video and give it a try.
```

[9]:
```
# Pruebo con mi solución

df_late_finishers = df[pd.to_datetime(df['assignment1_submission']) >= '2016']

print(len(df_late_finishers))
print('Total= early_finishers + late_finishers =', len(df_late_finishers)+␣
↪len(early_finishers))

df_late_finishers.head()
```

```
1056
Total= early_finishers + late_finishers = 2315
```

[9]:
```
                             student_id  assignment1_grade  \
2  D0F62040-CEB0-904C-F563-2F8620916C4E          85.512541
3  FFDF2B2C-F514-EF7F-6538-A6A53518E9DC          86.030665
6  3217BE3F-E4B0-C3B6-9F64-462456819CE4          87.498744
7  F1CB5AA1-B3DE-5460-FAFF-BE951FD38B5F          80.576090
9  E2C617C2-4654-622C-AB50-1550C4BE42A0          59.270882

         assignment1_submission  assignment2_grade  \
2  2016-01-09 05:36:02.389000000          85.512541
3  2016-04-30 06:50:39.801000000          68.824532
6  2016-03-05 11:05:25.408000000          69.998995
7  2016-01-24 18:24:25.619000000          72.518481
9  2016-03-06 12:06:26.185000000          59.270882

         assignment2_submission  assignment3_grade  \
2  2016-01-09 06:39:44.416000000          68.410033
3  2016-04-30 17:20:38.727000000          61.942079
6  2016-03-09 07:29:52.405000000          55.999196
7  2016-01-27 13:37:12.943000000          65.266633
9  2016-03-13 02:07:25.289000000          53.343794

         assignment3_submission  assignment4_grade  \
2  2016-01-15 20:22:45.882000000          54.728026
3  2016-05-12 07:47:16.326000000          49.553663
6  2016-03-16 22:31:24.316000000          50.399276
7  2016-01-30 14:34:36.581000000          65.266633
9  2016-03-17 07:30:09.241000000          53.343794

         assignment4_submission  assignment5_grade  \
```

```
2  2016-01-11 12:41:50.749000000          49.255224
3  2016-05-07 16:09:20.485000000          49.553663
6  2016-03-18 07:19:26.032000000          45.359349
7  2016-02-03 22:08:49.002000000          65.266633
9  2016-03-20 21:45:56.229000000          42.675035

          assignment5_submission  assignment6_grade  \
2  2016-01-11 17:31:12.489000000          44.329701
3  2016-05-24 12:51:18.016000000          44.598297
6  2016-03-19 10:35:41.869000000          45.359349
7  2016-02-16 14:22:23.664000000          65.266633
9  2016-03-27 15:55:04.414000000          38.407532

          assignment6_submission
2  2016-01-17 16:24:42.765000000
3  2016-05-26 08:09:12.058000000
6  2016-03-23 14:02:00.987000000
7  2016-02-18 08:35:04.796000000
9  2016-03-30 20:33:13.554000000
```

[35]:

```
# Here's my solution. First, the dataframe df and the early_finishers share␣
↪index values, so I really just
# want everything in the df which is not in early_finishers


# El símbolo "~" lo que hace es cambiar los True->False y los False->True
# Es una forma rápida de invertir la selección
# Por ende está invirtiendo los resultados, cambia los index de early_finishers␣
↪de True->False y todos los demás índices,
# o sea, aquellos que terminaron después los cambia de False->True
# Como resultado se obtienen los late_finishers a partir de elegir todos los␣
↪demás índices que son pertenecen a la
# df early_finishers
late_finishers=df[~df.index.isin(early_finishers.index)]
late_finishers.head()
```

[35]:

```
                              student_id  assignment1_grade  \
2  D0F62040-CEB0-904C-F563-2F8620916C4E          85.512541
3  FFDF2B2C-F514-EF7F-6538-A6A53518E9DC          86.030665
6  3217BE3F-E4B0-C3B6-9F64-462456819CE4          87.498744
7  F1CB5AA1-B3DE-5460-FAFF-BE951FD38B5F          80.576090
9  E2C617C2-4654-622C-AB50-1550C4BE42A0          59.270882

          assignment1_submission  assignment2_grade  \
2  2016-01-09 05:36:02.389000000          85.512541
3  2016-04-30 06:50:39.801000000          68.824532
6  2016-03-05 11:05:25.408000000          69.998995
7  2016-01-24 18:24:25.619000000          72.518481
```

```
9  2016-03-06 12:06:26.185000000         59.270882

          assignment2_submission  assignment3_grade  \
2  2016-01-09 06:39:44.416000000         68.410033
3  2016-04-30 17:20:38.727000000         61.942079
6  2016-03-09 07:29:52.405000000         55.999196
7  2016-01-27 13:37:12.943000000         65.266633
9  2016-03-13 02:07:25.289000000         53.343794

          assignment3_submission  assignment4_grade  \
2  2016-01-15 20:22:45.882000000         54.728026
3  2016-05-12 07:47:16.326000000         49.553663
6  2016-03-16 22:31:24.316000000         50.399276
7  2016-01-30 14:34:36.581000000         65.266633
9  2016-03-17 07:30:09.241000000         53.343794

          assignment4_submission  assignment5_grade  \
2  2016-01-11 12:41:50.749000000         49.255224
3  2016-05-07 16:09:20.485000000         49.553663
6  2016-03-18 07:19:26.032000000         45.359349
7  2016-02-03 22:08:49.002000000         65.266633
9  2016-03-20 21:45:56.229000000         42.675035

          assignment5_submission  assignment6_grade  \
2  2016-01-11 17:31:12.489000000         44.329701
3  2016-05-24 12:51:18.016000000         44.598297
6  2016-03-19 10:35:41.869000000         45.359349
7  2016-02-16 14:22:23.664000000         65.266633
9  2016-03-27 15:55:04.414000000         38.407532

          assignment6_submission
2  2016-01-17 16:24:42.765000000
3  2016-05-26 08:09:12.058000000
6  2016-03-23 14:02:00.987000000
7  2016-02-18 08:35:04.796000000
9  2016-03-30 20:33:13.554000000
```

[8]:
```
# There are lots of other ways to do this. For instance, you could just copy
 ↪and paste the first projection
# and change the sign from less than to greater than or equal to. This is ok,
 ↪but if you decide you want to
# change the date down the road you have to remember to change it in two places.
 ↪ You could also do a join of
# the dataframe df with early_finishers - if you do a left join you only keep
 ↪the items in the left dataframe,
# so this would have been a good answer. You also could have written a function
 ↪that determines if someone is
```

```
# early or late, and then called .apply() on the dataframe and added a new␣
↪column to the dataframe. This is a
# pretty reasonable answer as well.
```

[47]:
```
# Otra forma haciendo un merge

# Creo una merge y le agrega una columna '_merge' que indica de que matriz␣
↪vienen los datos: 'left_only', 'right_only' o 'both'
late_finishers2 = pd.merge(df, early_finishers, how='left', left_index=True,␣
↪right_index=True, indicator=True)

# Filtro los datos que solo vienen de la matriz df y excluyo lo de␣
↪early_finishers
# Además sólo me quedo con las columnas de df porque merge sumó las columnas de␣
↪early_finishers
late_finishers2 = late_finishers2[late_finishers2["_merge"]=='left_only'].iloc[:
↪,0:13]

# Cambio el nombre de las columnas por sus nombre originales porque el merge␣
↪les agrego _x
late_finishers2.columns = df.columns

# Corroboro que este método sea igual al del docente
(late_finishers2 == late_finishers).head()
```

[47]:
```
   student_id  assignment1_grade  assignment1_submission  assignment2_grade  \
2        True               True                    True               True
3        True               True                    True               True
6        True               True                    True               True
7        True               True                    True               True
9        True               True                    True               True

   assignment2_submission  assignment3_grade  assignment3_submission  \
2                    True               True                    True
3                    True               True                    True
6                    True               True                    True
7                    True               True                    True
9                    True               True                    True

   assignment4_grade  assignment4_submission  assignment5_grade  \
2               True                    True               True
3               True                    True               True
6               True                    True               True
7               True                    True               True
9               True                    True               True

   assignment5_submission  assignment6_grade  assignment6_submission
```

```
2                    True                 True                     True
3                    True                 True                     True
6                    True                 True                     True
7                    True                 True                     True
9                    True                 True                     True
```

[15]:
```python
# As you've seen, the pandas data frame object has a variety of statistical␣
↪functions associated with it. If
# we call the mean function directly on the data frame, we see that each of the␣
↪means for the assignments are
# calculated. Let's compare the means for our two populations

print(early_finishers['assignment1_grade'].mean())
print(late_finishers['assignment1_grade'].mean())
```

```
74.94728457024303
74.0450648477065
```

[50]:
```python
# Ok, these look pretty similar. But, are they the same? What do we mean by␣
↪similar? This is where the
# students' t-test comes in. It allows us to form the alternative hypothesis␣
↪("These are different") as well
# as the null hypothesis ("These are the same") and then test that null␣
↪hypothesis.

# When doing hypothesis testing, we have to choose a significance level as a␣
↪threshold for how much of a
# chance we're willing to accept. This significance level is typically called␣
↪alpha. #For this example, let's
# use a threshold of 0.05 for our alpha or 5%. Now this is a commonly used␣
↪number but it's really quite
# arbitrary.

# The SciPy library contains a number of different statistical tests and forms␣
↪a basis for hypothesis testing
# in Python and we're going to use the ttest_ind() function which does an␣
↪independent t-test (meaning the
# populations are not related to one another). The result of ttest_index() are␣
↪the t-statistic and a p-value.
# It's this latter value, the probability, which is most important to us, as it␣
↪indicates the chance (between
# 0 and 1) of our null hypothesis being True.

# Let's bring in our ttest_ind function
from scipy.stats import ttest_ind
```

```
# Let's run this function with our two populations, looking at the assignment 1␣
↪grades
ttest_ind(early_finishers['assignment1_grade'],␣
↪late_finishers['assignment1_grade'])
```

[50]: Ttest_indResult(statistic=1.322354085372139, pvalue=0.1861810110171455)

[51]:
```
# So here we see that the probability is 0.18, and this is above our alpha␣
↪value of 0.05. This means that we
# cannot reject the null hypothesis. The null hypothesis was that the two␣
↪populations are the same, and we
# don't have enough certainty in our evidence (because it is greater than␣
↪alpha) to come to a conclusion to
# the contrary. This doesn't mean that we have proven the populations are the␣
↪same.
```

[52]:
```
# Why don't we check the other assignment grades?
print(ttest_ind(early_finishers['assignment2_grade'],␣
↪late_finishers['assignment2_grade']))
print(ttest_ind(early_finishers['assignment3_grade'],␣
↪late_finishers['assignment3_grade']))
print(ttest_ind(early_finishers['assignment4_grade'],␣
↪late_finishers['assignment4_grade']))
print(ttest_ind(early_finishers['assignment5_grade'],␣
↪late_finishers['assignment5_grade']))
print(ttest_ind(early_finishers['assignment6_grade'],␣
↪late_finishers['assignment6_grade']))
```

```
Ttest_indResult(statistic=1.2514717608216366, pvalue=0.2108889627004424)
Ttest_indResult(statistic=1.6133726558705392, pvalue=0.10679998102227865)
Ttest_indResult(statistic=0.049671157386456125, pvalue=0.960388729789337)
Ttest_indResult(statistic=-0.05279315545404755, pvalue=0.9579012739746492)
Ttest_indResult(statistic=-0.11609743352612056, pvalue=0.9075854011989656)
```

[13]:
```
# Ok, so it looks like in this data we do not have enough evidence to suggest␣
↪the populations differ with
# respect to grade. Let's take a look at those p-values for a moment though,␣
↪because they are saying things
# that can inform experimental design down the road. For instance, one of the␣
↪assignments, assignment 3, has a
# p-value around 0.1. This means that if we accepted a level of chance␣
↪similarity of 11% this would have been
# considered statistically significant. As a research, this would suggest to me␣
↪that there is something here
# worth considering following up on. For instance, if we had a small number of␣
↪participants (we don't) or if
# there was something unique about this assignment as it relates to our␣
↪experiment (whatever it was) then
```

```
# there may be followup experiments we could run.
```

[106]:
```
# P-values have come under fire recently for being insuficient for telling us␣
↪enough about the interactions
# which are happening, and two other techniques, confidence intervalues and␣
↪bayesian analyses, are being used
# more regularly. One issue with p-values is that as you run more tests you are␣
↪likely to get a value which
# is statistically significant just by chance.

# Lets see a simulation of this. First, lets create a data frame of 100␣
↪columns, each with 100 numbers

# Mediante list comprehension crea 100 arrays con 100 nros al azar con valores␣
↪entre 0 y 1
# np.random.random(filas) estará definiendo el nro de filas de nuestra␣
↪dataframe
# range(columnas) estará definiendo el nro de columnas que queremos en nuestra␣
↪dataframe

## Recordar ## np.random.random() devuelve una distribución normal de los datos
df1=pd.DataFrame([np.random.random(100) for x in range(100)])
df1.head()
```

[106]:
```
         0         1         2         3         4         5         6   \
0  0.594408  0.855673  0.329023  0.100390  0.085777  0.091838  0.967431
1  0.394005  0.488888  0.379382  0.130293  0.805803  0.510559  0.042683
2  0.479941  0.924364  0.329247  0.167980  0.976703  0.295564  0.994833
3  0.390145  0.750313  0.907192  0.431511  0.014478  0.308244  0.515652
4  0.313648  0.252260  0.764231  0.182285  0.998335  0.712335  0.350913

         7         8         9   ...        90        91        92        93  \
0  0.614830  0.277144  0.647518  ...  0.241524  0.240630  0.958004  0.520959
1  0.563708  0.076590  0.529521  ...  0.071739  0.272692  0.231198  0.863647
2  0.654742  0.592719  0.250842  ...  0.421769  0.755745  0.832994  0.000862
3  0.817197  0.860680  0.597290  ...  0.165816  0.562646  0.804656  0.623111
4  0.044853  0.029193  0.878067  ...  0.763197  0.875019  0.647937  0.008770

          94        95        96        97        98        99
0  0.229199  0.633647  0.051517  0.282503  0.746843  0.097489
1  0.543649  0.209704  0.382704  0.574191  0.490194  0.111340
2  0.286140  0.461161  0.425977  0.772925  0.832434  0.888434
3  0.026347  0.419039  0.790155  0.439607  0.240447  0.052882
4  0.846934  0.640946  0.076911  0.635895  0.814370  0.104397

[5 rows x 100 columns]
```

[107]:
```
# Pause this and reflect -- do you understand the list comprehension and how I␣
↪created this DataFrame? You
# don't have to use a list comprehension to do this, but you should be able to␣
↪read this and figure out how it
# works as this is a commonly used approach on web forums.
```

[108]:
```
# Ok, let's create a second dataframe
df2=pd.DataFrame([np.random.random(100) for x in range(100)])
```

[109]:
```
df1.columns
```

[109]:
```
RangeIndex(start=0, stop=100, step=1)
```

[114]:
```
# Ahora tenemos 2 dataframes de 100x100 con datos al azar

# Qué pasaría si comparamos cada una de las columnas de una df con las mismas␣
↪cols de la otra data frame?
# Habrá diferencia estadísticamente significativa?
# La teoría dice que al ser datos al azar NO debería existir diferencias␣
↪esadísticamente significativas, pero por azar
# podría ser que haya.
# Para eso fijamos un valor de alpha, que es hasta cuánto azar nos permitimos␣
↪tolerar
# Alpha define la posibilidad de falsos positivos


# Are these two DataFrames the same? Maybe a better question is, for a given␣
↪row inside of df1, is it the same
# as the row inside df2?

# Let's take a look. Let's say our critical value is 0.1, or and alpha of 10%.␣
↪And we're going to compare each
# column in df1 to the same numbered column in df2. And we'll report when the␣
↪p-value is less than 10%,
# which means that we have sufficient evidence to say that the columns between␣
↪df1 and df2 are different.

# Let's write this in a function called test_columns
def test_columns(alpha=0.1):
    # I want to keep track of how many differ
    num_diff=0
    # And now we can just iterate over the columns
    for col in df1.columns:
        # we can run out ttest_ind between the two dataframes
        teststat,pval=ttest_ind(df1[col],df2[col])
        # and we check the pvalue versus the alpha
        if pval<=alpha:
```

```
            # And now we'll just print out if they are different and increment␣
↪the num_diff
            print("Col {} is statistically significantly different at alpha={},␣
↪pval={}".format(col,alpha,pval))
            num_diff=num_diff+1
    # and let's print out some summary stats
    print("Total number different was {}, which is {}%".
↪format(num_diff,float(num_diff)/len(df1.columns)*100))

# And now lets actually run this
test_columns()
```

```
Col 13 is statistically significantly different at alpha=0.1,
pval=0.07130524662924308
Col 24 is statistically significantly different at alpha=0.1,
pval=0.04028335043864821
Col 30 is statistically significantly different at alpha=0.1,
pval=0.056934084967090674
Col 41 is statistically significantly different at alpha=0.1,
pval=0.09107532448808293
Col 59 is statistically significantly different at alpha=0.1,
pval=0.0088755258985562
Col 62 is statistically significantly different at alpha=0.1,
pval=0.006634461844164361
Col 64 is statistically significantly different at alpha=0.1,
pval=0.008350383533474197
Col 75 is statistically significantly different at alpha=0.1,
pval=0.06781893833323978
Col 84 is statistically significantly different at alpha=0.1,
pval=0.03161045801868621
Col 94 is statistically significantly different at alpha=0.1,
pval=0.06796430599349267
Col 97 is statistically significantly different at alpha=0.1,
pval=0.03363468194559633
Total number different was 11, which is 11.0%
```

[111]:
```
# Interesting, so we see that there are a bunch of columns that are different!␣
↪In fact, that number looks a
# lot like the alpha value we chose. So what's going on - shouldn't all of the␣
↪columns be the same? Remember
# that all the ttest does is check if two sets are similar given some level of␣
↪confidence, in our case, 10%.
# The more random comparisons you do, the more will just happen to be the same␣
↪by chance. In this example, we
# checked 100 columns, so we would expect there to be roughly 10 of them if our␣
↪alpha was 0.1.
```

```
# We can test some other alpha values as well
test_columns(0.05)
```

```
Col 24 is statistically significantly different at alpha=0.05,
pval=0.04028335043864821
Col 59 is statistically significantly different at alpha=0.05,
pval=0.0088755258985562
Col 62 is statistically significantly different at alpha=0.05,
pval=0.006634461844164361
Col 64 is statistically significantly different at alpha=0.05,
pval=0.008350383533474197
Col 84 is statistically significantly different at alpha=0.05,
pval=0.03161045801868621
Col 97 is statistically significantly different at alpha=0.05,
pval=0.03363468194559633
Total number different was 6, which is 6.0%
```

[112]:
```
test_columns(0.01)
```

```
Col 59 is statistically significantly different at alpha=0.01,
pval=0.0088755258985562
Col 62 is statistically significantly different at alpha=0.01,
pval=0.006634461844164361
Col 64 is statistically significantly different at alpha=0.01,
pval=0.008350383533474197
Total number different was 3, which is 3.0%
```

[117]:
```
# Esto es lo mismo que la linea 112
teststat,pval = ttest_ind(df1,df2,axis=0)
a= pval
a<0.01
```

[117]:
```
array([False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False,  True, False, False,  True,
       False,  True, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False])
```

[104]:
```
# So, keep this in mind when you are doing statistical tests like the t-test␣
↪which has a p-value. Understand
```

```
# that this p-value isn't magic, that it's a threshold for you when reporting␣
 ↪results and trying to answer
# your hypothesis. What's a reasonable threshold? Depends on your question, and␣
 ↪you need to engage domain
# experts to better understand what they would consider significant.

# Just for fun, lets recreate that second dataframe using a non-normal␣
 ↪distribution, I'll arbitrarily chose
# chi squared
df2=pd.DataFrame([np.random.chisquare(df=1,size=100) for x in range(100)])
test_columns()
```

```
Col 0 is statistically significantly different at alpha=0.1,
pval=4.440349113793565e-06
Col 1 is statistically significantly different at alpha=0.1,
pval=0.0008278680435964049
Col 2 is statistically significantly different at alpha=0.1,
pval=0.003816124759556187
Col 3 is statistically significantly different at alpha=0.1,
pval=0.0004911934961255687
Col 4 is statistically significantly different at alpha=0.1,
pval=0.039860850687161185
Col 5 is statistically significantly different at alpha=0.1,
pval=2.2555734402440538e-06
Col 6 is statistically significantly different at alpha=0.1,
pval=0.0008535080039121404
Col 7 is statistically significantly different at alpha=0.1,
pval=0.00029966317456492266
Col 8 is statistically significantly different at alpha=0.1,
pval=0.0015915978713177514
Col 9 is statistically significantly different at alpha=0.1,
pval=9.538700923844107e-05
Col 10 is statistically significantly different at alpha=0.1,
pval=0.013064353749202805
Col 11 is statistically significantly different at alpha=0.1,
pval=5.744477730023852e-05
Col 12 is statistically significantly different at alpha=0.1,
pval=0.014172222783799125
Col 13 is statistically significantly different at alpha=0.1,
pval=0.028685770582321753
Col 14 is statistically significantly different at alpha=0.1,
pval=0.0002461498709477624
Col 15 is statistically significantly different at alpha=0.1,
pval=0.0002816836426508307
Col 16 is statistically significantly different at alpha=0.1,
pval=0.0001076662126999714
Col 17 is statistically significantly different at alpha=0.1,
```

```
pval=0.001646232090858095
Col 18 is statistically significantly different at alpha=0.1,
pval=0.000445734269804215
Col 19 is statistically significantly different at alpha=0.1,
pval=9.877432449027482e-05
Col 20 is statistically significantly different at alpha=0.1,
pval=0.00019681335732241656
Col 21 is statistically significantly different at alpha=0.1,
pval=0.009164795633881808
Col 22 is statistically significantly different at alpha=0.1,
pval=9.058192344443374e-06
Col 23 is statistically significantly different at alpha=0.1,
pval=0.0018816893544193933
Col 24 is statistically significantly different at alpha=0.1,
pval=0.0009825311132196984
Col 25 is statistically significantly different at alpha=0.1,
pval=0.0012357801685886315
Col 26 is statistically significantly different at alpha=0.1,
pval=0.0002435334974719032
Col 27 is statistically significantly different at alpha=0.1,
pval=0.002252618585156726
Col 28 is statistically significantly different at alpha=0.1,
pval=0.00014698508947752741
Col 29 is statistically significantly different at alpha=0.1,
pval=0.0008411081213415994
Col 30 is statistically significantly different at alpha=0.1,
pval=0.005548460257937831
Col 31 is statistically significantly different at alpha=0.1,
pval=0.0025424232241385417
Col 32 is statistically significantly different at alpha=0.1,
pval=0.002292819641364267
Col 33 is statistically significantly different at alpha=0.1,
pval=0.007427615295907003
Col 34 is statistically significantly different at alpha=0.1,
pval=2.712546505387272e-06
Col 35 is statistically significantly different at alpha=0.1,
pval=2.4636092512286185e-05
Col 36 is statistically significantly different at alpha=0.1,
pval=0.004216967150845883
Col 37 is statistically significantly different at alpha=0.1,
pval=8.026041014649346e-05
Col 38 is statistically significantly different at alpha=0.1,
pval=6.821052530695058e-05
Col 39 is statistically significantly different at alpha=0.1,
pval=0.0006452064105905297
Col 40 is statistically significantly different at alpha=0.1,
pval=0.0013628430719404086
Col 41 is statistically significantly different at alpha=0.1,
```

```
pval=0.030528775505445017
Col 42 is statistically significantly different at alpha=0.1,
pval=0.00033218084234707583
Col 43 is statistically significantly different at alpha=0.1,
pval=0.006285323084813953
Col 44 is statistically significantly different at alpha=0.1,
pval=0.000524058229319966
Col 45 is statistically significantly different at alpha=0.1,
pval=0.005131297771824898
Col 46 is statistically significantly different at alpha=0.1,
pval=5.8188364077347906e-05
Col 47 is statistically significantly different at alpha=0.1,
pval=0.0016906251217854008
Col 48 is statistically significantly different at alpha=0.1,
pval=0.012654872339779785
Col 49 is statistically significantly different at alpha=0.1,
pval=0.0013543754789274295
Col 50 is statistically significantly different at alpha=0.1,
pval=8.02016301727056e-05
Col 51 is statistically significantly different at alpha=0.1,
pval=0.0002654036500627568
Col 52 is statistically significantly different at alpha=0.1,
pval=0.002698336979172998
Col 53 is statistically significantly different at alpha=0.1,
pval=2.2973554464482796e-05
Col 54 is statistically significantly different at alpha=0.1,
pval=4.181962997433052e-06
Col 55 is statistically significantly different at alpha=0.1,
pval=1.2699562835432457e-06
Col 56 is statistically significantly different at alpha=0.1,
pval=0.0006235653582811482
Col 57 is statistically significantly different at alpha=0.1,
pval=0.007144104238854865
Col 58 is statistically significantly different at alpha=0.1,
pval=0.00017993884141847663
Col 59 is statistically significantly different at alpha=0.1,
pval=0.0001023697184619525
Col 60 is statistically significantly different at alpha=0.1,
pval=0.017027905640924632
Col 61 is statistically significantly different at alpha=0.1,
pval=0.001188727791817188
Col 62 is statistically significantly different at alpha=0.1,
pval=0.0005230655965326533
Col 63 is statistically significantly different at alpha=0.1,
pval=0.0005747303711462303
Col 64 is statistically significantly different at alpha=0.1,
pval=0.0023461771390345574
Col 65 is statistically significantly different at alpha=0.1,
```

```
pval=0.000645320626752544
Col 66 is statistically significantly different at alpha=0.1,
pval=4.770104783222682e-05
Col 67 is statistically significantly different at alpha=0.1,
pval=1.863550891713227e-05
Col 68 is statistically significantly different at alpha=0.1,
pval=0.002645756077512991
Col 69 is statistically significantly different at alpha=0.1,
pval=0.04405540391461409
Col 70 is statistically significantly different at alpha=0.1,
pval=0.0002507015360888315
Col 71 is statistically significantly different at alpha=0.1,
pval=0.0010400119692073001
Col 72 is statistically significantly different at alpha=0.1,
pval=6.665016845409382e-05
Col 73 is statistically significantly different at alpha=0.1,
pval=9.552645812564957e-07
Col 74 is statistically significantly different at alpha=0.1,
pval=0.03024161344241096
Col 75 is statistically significantly different at alpha=0.1,
pval=0.00028235848356012664
Col 76 is statistically significantly different at alpha=0.1,
pval=0.003059428934244056
Col 77 is statistically significantly different at alpha=0.1,
pval=0.00015135717708840898
Col 78 is statistically significantly different at alpha=0.1,
pval=0.0002622632745055216
Col 79 is statistically significantly different at alpha=0.1,
pval=0.02091922399653835
Col 80 is statistically significantly different at alpha=0.1,
pval=0.0066116259819389185
Col 81 is statistically significantly different at alpha=0.1,
pval=0.00040291904114427485
Col 82 is statistically significantly different at alpha=0.1,
pval=0.005669347580927994
Col 83 is statistically significantly different at alpha=0.1,
pval=7.997843781252031e-05
Col 84 is statistically significantly different at alpha=0.1,
pval=0.007253928753058471
Col 85 is statistically significantly different at alpha=0.1,
pval=0.008842879173439639
Col 86 is statistically significantly different at alpha=0.1,
pval=3.0434046615969643e-05
Col 87 is statistically significantly different at alpha=0.1,
pval=0.00032513813461845727
Col 88 is statistically significantly different at alpha=0.1,
pval=0.0006265369749005545
Col 89 is statistically significantly different at alpha=0.1,
```

```
pval=0.003652517551300469
Col 90 is statistically significantly different at alpha=0.1,
pval=0.00022956879896485907
Col 91 is statistically significantly different at alpha=0.1,
pval=0.026155059258430403
Col 92 is statistically significantly different at alpha=0.1,
pval=0.004153977112149194
Col 93 is statistically significantly different at alpha=0.1,
pval=0.00026970780504012706
Col 94 is statistically significantly different at alpha=0.1,
pval=7.426560610627656e-05
Col 95 is statistically significantly different at alpha=0.1,
pval=0.008584743695875271
Col 96 is statistically significantly different at alpha=0.1,
pval=0.019008823844927785
Col 97 is statistically significantly different at alpha=0.1,
pval=0.0030166414173591885
Col 98 is statistically significantly different at alpha=0.1,
pval=0.00040838412445619977
Col 99 is statistically significantly different at alpha=0.1,
pval=0.0003224913508063252
Total number different was 100, which is 100.0%
```

[20]:
```
# Now we see that all or most columns test to be statistically significant at
↪the 10% level.
```

In this lecture, we've discussed just some of the basics of hypothesis testing in Python. I introduced you to the SciPy library, which you can use for the students t test. We've discussed some of the practical issues which arise from looking for statistical significance. There's much more to learn about hypothesis testing, for instance, there are different tests used, depending on the shape of your data and different ways to report results instead of just p-values such as confidence intervals or bayesian analyses. But this should give you a basic idea of where to start when comparing two populations for differences, which is a common task for data scientists.