BasicStatisticalTesting

November 15, 2021

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]:
```python
# 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]:
```python
# 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
```

[3]:
```python
# 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()
```

[3]:

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

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

[5]:
```
# 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[pd.to_datetime(df['assignment1_submission']) < '2016']
early_finishers.head()
```

[5]:
```
                             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
```

[6]:
```
# 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.
```

[7]:
```
# 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
late_finishers=df[~df.index.isin(early_finishers.index)]
late_finishers.head()
```

[7]:
```
                            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.
```

[9]:
```
# 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
```

[10]:
```
# 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'])
```

[10]:
```
Ttest_indResult(statistic=1.322354085372139, pvalue=0.1861810110171455)
```

[11]:
```
# 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.
```

[12]:
```
# 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.
```

[14]:
```
# 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
df1=pd.DataFrame([np.random.random(100) for x in range(100)])
df1.head()
```

[14]:

```
          0         1         2         3         4         5         6   \
0  0.249058  0.748065  0.245871  0.164790  0.321661  0.355034  0.882641
1  0.731003  0.592047  0.488109  0.969236  0.873551  0.976036  0.296337
2  0.315190  0.081075  0.326315  0.166250  0.257618  0.406736  0.996209
3  0.512193  0.639647  0.241200  0.819644  0.439495  0.111223  0.982239
4  0.506768  0.799246  0.233428  0.849028  0.682260  0.904570  0.962683

          7         8         9   ...        90        91        92        93  \
0  0.304377  0.409110  0.080049  ...  0.900937  0.740939  0.353839  0.928663
1  0.398915  0.258550  0.925547  ...  0.478983  0.242443  0.225005  0.803914
2  0.986399  0.339623  0.828867  ...  0.157851  0.496227  0.983652  0.971251
3  0.574194  0.935731  0.609363  ...  0.373206  0.271515  0.942992  0.701573
4  0.734952  0.933580  0.904864  ...  0.263317  0.895435  0.327082  0.872009

          94        95        96        97        98        99
0  0.011866  0.876878  0.609879  0.721572  0.589627  0.554882
1  0.576076  0.190479  0.696137  0.573191  0.505417  0.626049
2  0.986353  0.547430  0.255769  0.165918  0.242719  0.575260
3  0.039384  0.085479  0.284002  0.509064  0.112163  0.334849
4  0.566034  0.474804  0.258897  0.316417  0.433785  0.127601

[5 rows x 100 columns]
```

[15]:

```
# 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.
```

[16]:

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

[17]:

```
# 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 isn't less than 10%,
```

```
# which means that we have sufficient evidence to say that the columns 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 4 is statistically significantly different at alpha=0.1,
pval=0.03171904508748956
Col 9 is statistically significantly different at alpha=0.1,
pval=0.05535264960034999
Col 13 is statistically significantly different at alpha=0.1,
pval=0.06411026546897207
Col 25 is statistically significantly different at alpha=0.1,
pval=0.06474837389971169
Col 46 is statistically significantly different at alpha=0.1,
pval=0.0766378775191977
Col 73 is statistically significantly different at alpha=0.1,
pval=0.021821747279110435
Col 76 is statistically significantly different at alpha=0.1,
pval=0.04067259983835964
Col 85 is statistically significantly different at alpha=0.1,
pval=0.0029530262143926777
Col 86 is statistically significantly different at alpha=0.1,
pval=0.010704932840894129
Total number different was 9, which is 9.0%
```

[18]:
```
# 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 4 is statistically significantly different at alpha=0.05,
pval=0.03171904508748956
Col 73 is statistically significantly different at alpha=0.05,
pval=0.021821747279110435
Col 76 is statistically significantly different at alpha=0.05,
pval=0.04067259983835964
Col 85 is statistically significantly different at alpha=0.05,
pval=0.0029530262143926777
Col 86 is statistically significantly different at alpha=0.05,
pval=0.010704932840894129
Total number different was 5, which is 5.0%
```

[19]:
```
# 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=0.0006359131132062404
Col 1 is statistically significantly different at alpha=0.1,
pval=1.6254596067473314e-05
Col 2 is statistically significantly different at alpha=0.1,
pval=0.032780760709076824
Col 3 is statistically significantly different at alpha=0.1,
pval=0.0032178974546995965
```

```
Col 4 is statistically significantly different at alpha=0.1,
pval=0.0433143837061582
Col 5 is statistically significantly different at alpha=0.1,
pval=7.931565914508593e-05
Col 6 is statistically significantly different at alpha=0.1,
pval=0.0017013488226864455
Col 7 is statistically significantly different at alpha=0.1,
pval=0.00035664966978204754
Col 8 is statistically significantly different at alpha=0.1,
pval=0.0008750262277182237
Col 9 is statistically significantly different at alpha=0.1,
pval=3.9005870973813224e-05
Col 10 is statistically significantly different at alpha=0.1,
pval=0.0001340016519264235
Col 11 is statistically significantly different at alpha=0.1,
pval=2.9387732963990273e-06
Col 12 is statistically significantly different at alpha=0.1,
pval=0.0002593817469470156
Col 13 is statistically significantly different at alpha=0.1,
pval=0.0001639697026470583
Col 14 is statistically significantly different at alpha=0.1,
pval=0.00015526373271627754
Col 15 is statistically significantly different at alpha=0.1,
pval=4.1837854872688554e-05
Col 16 is statistically significantly different at alpha=0.1,
pval=0.00139904683631816
Col 17 is statistically significantly different at alpha=0.1,
pval=0.008340935008253801
Col 18 is statistically significantly different at alpha=0.1,
pval=0.0005255784024108116
Col 19 is statistically significantly different at alpha=0.1,
pval=0.0010508383648962462
Col 20 is statistically significantly different at alpha=0.1,
pval=9.250384193688497e-05
Col 21 is statistically significantly different at alpha=0.1,
pval=0.0004139078346825869
Col 22 is statistically significantly different at alpha=0.1,
pval=0.000239886090894122
Col 23 is statistically significantly different at alpha=0.1,
pval=0.005114973931480402
Col 24 is statistically significantly different at alpha=0.1,
pval=3.9612287309026586e-06
Col 25 is statistically significantly different at alpha=0.1,
pval=0.001183662110645545
Col 26 is statistically significantly different at alpha=0.1,
pval=0.0013250467524673893
Col 27 is statistically significantly different at alpha=0.1,
pval=0.007141366239129l675
```

```
Col 28 is statistically significantly different at alpha=0.1,
pval=0.0018050525585905151
Col 29 is statistically significantly different at alpha=0.1,
pval=1.8443168650897388e-07
Col 30 is statistically significantly different at alpha=0.1,
pval=0.00018610780116814143
Col 31 is statistically significantly different at alpha=0.1,
pval=0.003288231435508567
Col 32 is statistically significantly different at alpha=0.1,
pval=0.00022788735453261414
Col 33 is statistically significantly different at alpha=0.1,
pval=0.0005420802253194416
Col 34 is statistically significantly different at alpha=0.1,
pval=0.00043337835781774326
Col 35 is statistically significantly different at alpha=0.1,
pval=0.0009296626942443332
Col 36 is statistically significantly different at alpha=0.1,
pval=0.015486530171592511
Col 37 is statistically significantly different at alpha=0.1,
pval=0.08968808146980235
Col 38 is statistically significantly different at alpha=0.1,
pval=0.0003984738782135047
Col 39 is statistically significantly different at alpha=0.1,
pval=0.00029812833313821313
Col 40 is statistically significantly different at alpha=0.1,
pval=0.00020580570179961344
Col 41 is statistically significantly different at alpha=0.1,
pval=0.016524830474479658
Col 42 is statistically significantly different at alpha=0.1,
pval=0.004414785565347055
Col 43 is statistically significantly different at alpha=0.1,
pval=0.0006514221896553882
Col 44 is statistically significantly different at alpha=0.1,
pval=0.0009246080966774699
Col 45 is statistically significantly different at alpha=0.1,
pval=3.947719385039122e-05
Col 46 is statistically significantly different at alpha=0.1,
pval=0.003936803652126271
Col 47 is statistically significantly different at alpha=0.1,
pval=0.00021471514569468722
Col 48 is statistically significantly different at alpha=0.1,
pval=0.00021533428873488688
Col 49 is statistically significantly different at alpha=0.1,
pval=0.08198381088074698
Col 50 is statistically significantly different at alpha=0.1,
pval=0.0005833899282268582
Col 51 is statistically significantly different at alpha=0.1,
pval=0.0008976227172661998
```

```
Col 52 is statistically significantly different at alpha=0.1,
pval=0.002183481627605226
Col 53 is statistically significantly different at alpha=0.1,
pval=0.00011517331466172489
Col 54 is statistically significantly different at alpha=0.1,
pval=0.0004539298416324909
Col 55 is statistically significantly different at alpha=0.1,
pval=0.0005972354479418133
Col 56 is statistically significantly different at alpha=0.1,
pval=0.00039537371324602574
Col 57 is statistically significantly different at alpha=0.1,
pval=1.5992684064779836e-05
Col 58 is statistically significantly different at alpha=0.1,
pval=0.0035581119921681737
Col 59 is statistically significantly different at alpha=0.1,
pval=0.0009069465410284678
Col 60 is statistically significantly different at alpha=0.1,
pval=0.001490617627902622
Col 61 is statistically significantly different at alpha=0.1,
pval=9.982665694645061e-05
Col 62 is statistically significantly different at alpha=0.1,
pval=0.0001855201502830867
Col 63 is statistically significantly different at alpha=0.1,
pval=0.0717892199552874
Col 64 is statistically significantly different at alpha=0.1,
pval=0.00011111316914952978
Col 65 is statistically significantly different at alpha=0.1,
pval=0.0007326247241706332
Col 66 is statistically significantly different at alpha=0.1,
pval=0.0006985304647035251
Col 67 is statistically significantly different at alpha=0.1,
pval=4.043055942025079e-05
Col 68 is statistically significantly different at alpha=0.1,
pval=0.0030620341209672865
Col 69 is statistically significantly different at alpha=0.1,
pval=0.0002566485047083988
Col 70 is statistically significantly different at alpha=0.1,
pval=1.5265122338601147e-05
Col 71 is statistically significantly different at alpha=0.1,
pval=0.0025159633542264875
Col 72 is statistically significantly different at alpha=0.1,
pval=0.006683482780335058
Col 73 is statistically significantly different at alpha=0.1,
pval=0.0004996694763506213
Col 74 is statistically significantly different at alpha=0.1,
pval=0.00024394144210986108
Col 75 is statistically significantly different at alpha=0.1,
pval=7.077435724052419e-05
```

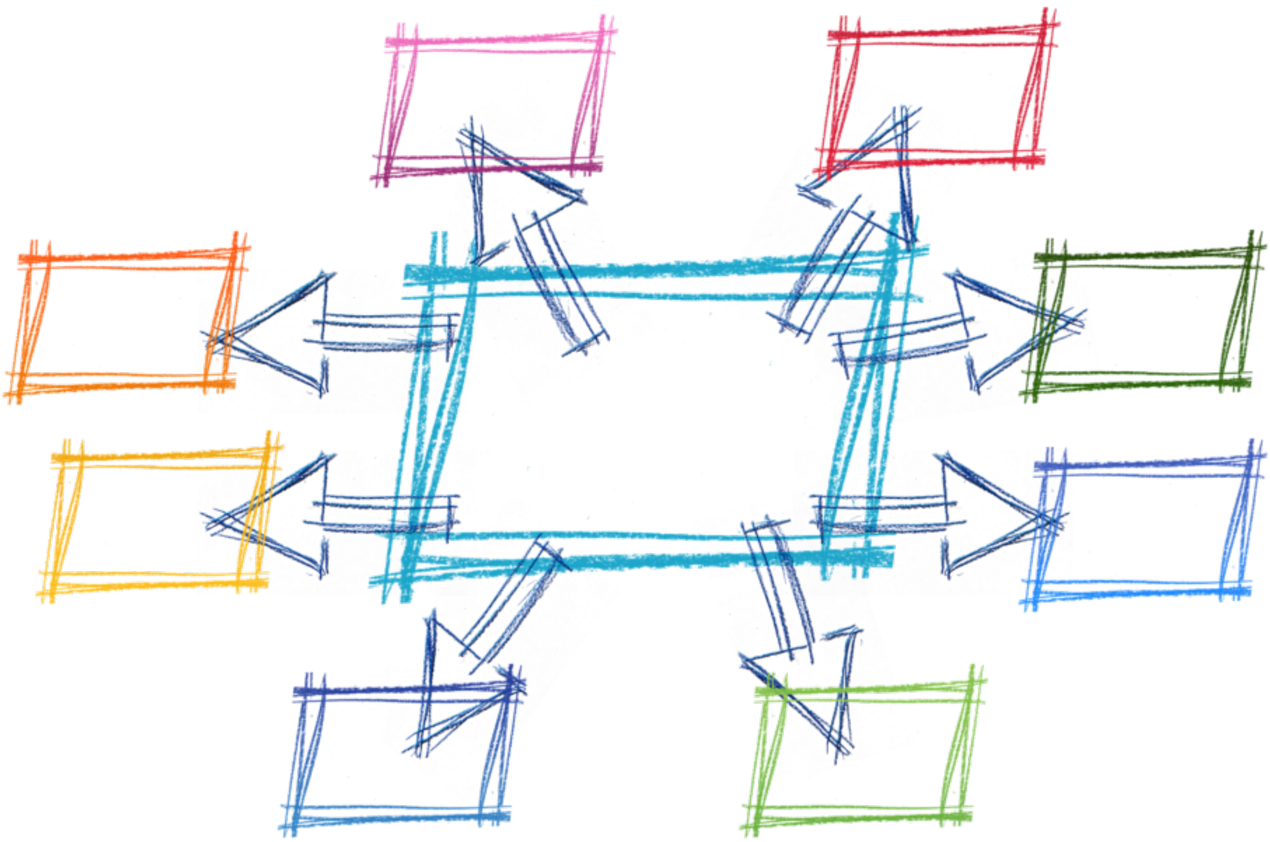
```
Col 76 is statistically significantly different at alpha=0.1,
pval=0.0074093508129584405
Col 77 is statistically significantly different at alpha=0.1,
pval=0.006900999886679228
Col 78 is statistically significantly different at alpha=0.1,
pval=0.0010484064032203217
Col 79 is statistically significantly different at alpha=0.1,
pval=0.0023846343634806146
Col 80 is statistically significantly different at alpha=0.1,
pval=8.180792170754975e-05
Col 81 is statistically significantly different at alpha=0.1,
pval=5.49471394287242e-05
Col 82 is statistically significantly different at alpha=0.1,
pval=0.00033420295079797965
Col 83 is statistically significantly different at alpha=0.1,
pval=0.003652590090290096
Col 84 is statistically significantly different at alpha=0.1,
pval=0.0005087442921717129
Col 85 is statistically significantly different at alpha=0.1,
pval=0.010877718451872447
Col 86 is statistically significantly different at alpha=0.1,
pval=0.006487775025921745
Col 87 is statistically significantly different at alpha=0.1,
pval=0.00012983278069724697
Col 88 is statistically significantly different at alpha=0.1,
pval=0.0003217680289398193
Col 89 is statistically significantly different at alpha=0.1,
pval=0.0007595005579977705
Col 90 is statistically significantly different at alpha=0.1,
pval=0.0009007314132232909
Col 91 is statistically significantly different at alpha=0.1,
pval=0.0041773223441325755
Col 92 is statistically significantly different at alpha=0.1,
pval=0.00013412252503388242
Col 93 is statistically significantly different at alpha=0.1,
pval=0.0038256024065565237
Col 94 is statistically significantly different at alpha=0.1,
pval=0.0025884060593381543
Col 95 is statistically significantly different at alpha=0.1,
pval=0.0008645653623897333
Col 96 is statistically significantly different at alpha=0.1,
pval=0.00033563002443083584
Col 97 is statistically significantly different at alpha=0.1,
pval=9.000223928968477e-06
Col 98 is statistically significantly different at alpha=0.1,
pval=0.007777985999968882
Col 99 is statistically significantly different at alpha=0.1,
pval=0.00024422963411248024
```

```
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.

towards data science



Data Scientists, The 5 Graph Algorithms that you should know

Rahul Ag

We as data scientists have gotten quite comfortable with Pandas or SQL or any other relational database.

We are used to seeing our users in rows with their attributes as columns. But does the real world really behave like that?

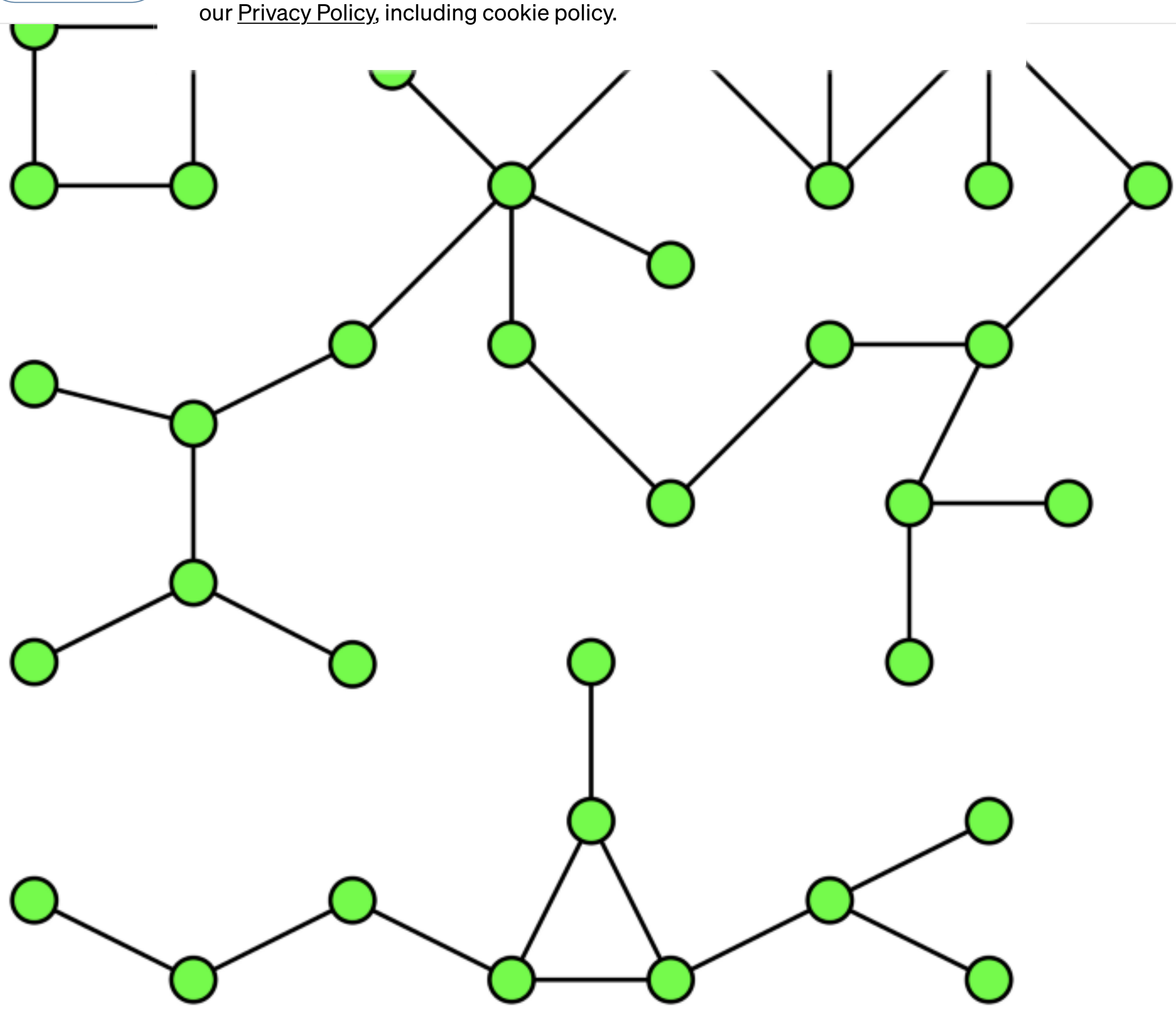
In a connected world, users cannot be considered as independent entities. They have got certain relationships between each other and we would sometimes like to include such relationships while building our machine learning models.

Now while in a relational database, we cannot use such relations between different rows(users), in a graph database it is fairly trivial to do that.

In this post, I am going to be talking about some of the most important graph algorithms you should know and how to implement them using Python.

Also, here is a Graph Analytics for Big Data course on Coursera by UCSanDiego which I highly recommend to learn the basics of graph theory.

1. Connected Components



A graph with 3 connected components

We all know how clustering works?

You can think of Connected Components in very layman's terms as a sort of a hard clustering algorithm which finds clusters/islands in related/connected data.

As a concrete example: ***Say you have data about roads joining any two cities in the world. And you need to find out all the continents in the world and which city they contain.***

How will you achieve that? Come on give some thought.

BFS/DFS. I get the code up and running using `Networkx`.

Applications

From a **Retail Perspective**: Let us say, we have a lot of customers using a lot of accounts. One way in which we can use the Connected components algorithm is to find out distinct families in our dataset.

We can assume edges(roads) between CustomerIDs based on same credit card usage, or same address or same mobile number, etc. Once we have those connections, we can then run the connected component algorithm on the same to create individual clusters to which we can then assign a family ID.

We can then use these family IDs to provide personalized recommendations based on family needs. We can also use this family ID to fuel our classification algorithms by creating grouped features based on family.

From a **Finance Perspective**: Another use case would be to capture fraud using these family IDs. If an account has done fraud in the past, it is highly probable that the connected accounts are also susceptible to fraud.

The possibilities are only limited by your own imagination.

Code

We will be using the `Networkx` module in Python for creating and analyzing our graphs.

Let us start with an example graph which we are using for our purpose. Contains cities and distance information between them.

Graph with Some random distances

We first start by creating a list of edges along with the distances which we will add as the weight of the edge:

```
edgelist = [['Mannheim', 'Frankfurt', 85], ['Mannheim', 'Karlsruhe',
80], ['Erfurt', 'Wurzburg', 186], ['Munchen', 'Numberg', 167],
['Munchen', 'Augsburg', 84], ['Munchen', 'Kassel', 502], ['Numberg',
'Stuttgart', 183], ['Numberg', 'Wurzburg', 103], ['Numberg',
'Munchen', 167], ['Stuttgart', 'Numberg', 183], ['Augsburg',
'Munchen', 84], ['Augsburg', 'Karlsruhe', 250], ['Kassel',
'Munchen', 502], ['Kassel', 'Frankfurt', 173], ['Frankfurt',
'Mannheim', 85], ['Frankfurt', 'Wurzburg', 217], ['Frankfurt',
'Kassel', 173], ['Wurzburg', 'Numberg', 103], ['Wurzburg', 'Erfurt',
186], ['Wurzburg', 'Frankfurt', 217], ['Karlsruhe', 'Mannheim', 80],
['Karlsruhe', 'Augsburg', 250],["Mumbai", "Delhi",400],["Delhi",
"Kolkata",500],["Kolkata", "Bangalore",600],["TX", "NY",1200],
["ALB", "NY",800]]
```

2. Shortest Path

```
g = nx.Graph()
```

',

nd

nted

above.

Continuing with the above example only, we are given a graph with the cities of Germany and the respective distance between them.

You want to find out how to go from Frankfurt (The starting node) to Munchen by covering the shortest distance.

The algorithm that we use for this problem is called **Dijkstra**. In Dijkstra's own words:

> *What is the shortest way to travel from Rotterdam to Groningen, in general: from given city to given city. It is the algorithm for the shortest path, which I designed in about twenty minutes. One morning I was shopping in Amsterdam with my young fiancée, and tired, we sat down on the café terrace to drink a cup of coffee and I was just thinking about whether I could do this, and I then designed the algorithm for the shortest path. As I said, it was a twenty-minute invention. In fact, it was published in '59, three years later. The publication*

> *without pen* *ıplexities. Eventually that algorithm became, to my great amazement, one of the cornerstones of my fame.*
>
> *— Edsger Dijkstra, in an interview with Philip L. Frana, Communications of the ACM, 2001[3]*

Applications

- Variations of the Dijkstra algorithm is used extensively in Google Maps to find the shortest routes.
- You are in a Walmart Store. You have different Aisles and distance between all the aisles. You want to provide the shortest pathway to the customer from Aisle A to Aisle D.

- You have seen how LinkedIn shows up 1st-degree connections, 2nd-degree connections. What goes on behind the scenes?

Code

```
print(nx.shortest_path(g, 'Stuttgart','Frankfurt',weight='weight'))
print(nx.shortest_path_length(g,
'Stuttgart','Frankfurt',weight='weight'))
---------------------------------------------------------
['Stuttgart', 'Numberg', 'Wurzburg', 'Frankfurt']
503
```

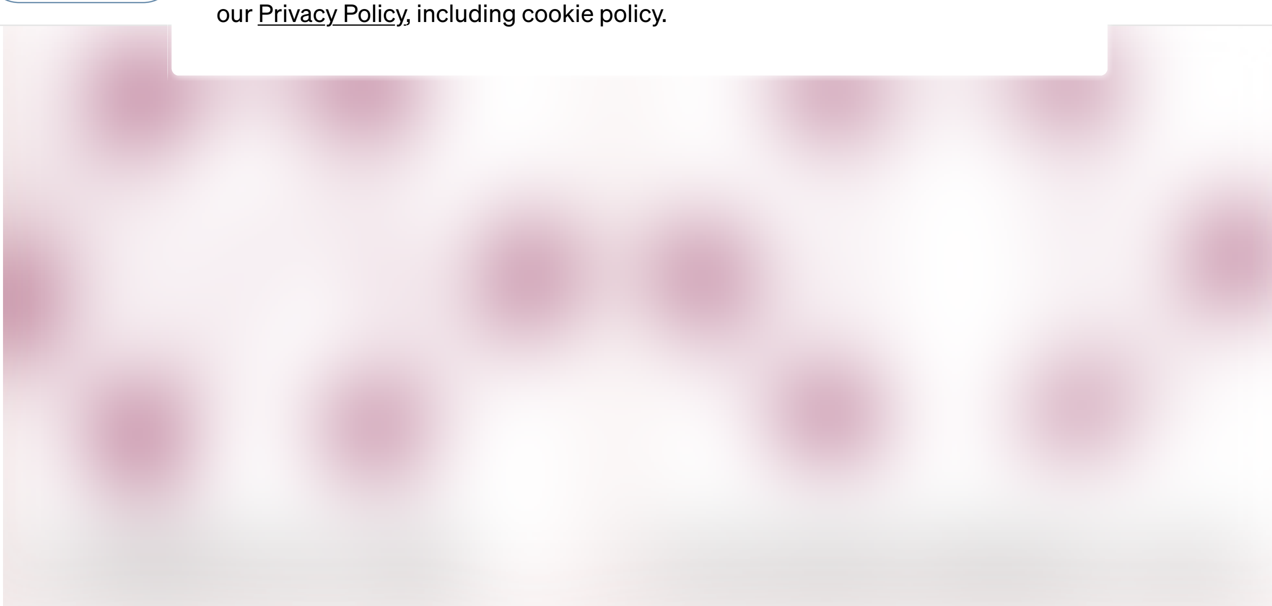
You can also find Shortest paths between all pairs using:

```
for x in nx.all_pairs_dijkstra_path(g,weight='weight'):
    print(x)
---------------------------------------------------------
```

```
'Frankfu
'Wurzburg'], 'Munchen': ['Mannheim', 'Karlsruhe', 'Augsburg',
'Munchen'], 'Erfurt': ['Mannheim', 'Frankfurt', 'Wurzburg',
'Erfurt'], 'Numberg': ['Mannheim', 'Frankfurt', 'Wurzburg',
'Numberg'], 'Stuttgart': ['Mannheim', 'Frankfurt', 'Wurzburg',
'Numberg', 'Stuttgart']})

('Frankfurt', {'Frankfurt': ['Frankfurt'], 'Mannheim': ['Frankfurt',
'Mannheim'], 'Kassel': ['Frankfurt', 'Kassel'], 'Wurzburg':
['Frankfurt', 'Wurzburg'], 'Karlsruhe': ['Frankfurt', 'Mannheim',
'Karlsruhe'], 'Augsburg': ['Frankfurt', 'Mannheim', 'Karlsruhe',
'Augsburg'], 'Munchen': ['Frankfurt', 'Wurzburg', 'Numberg',
'Munchen'], 'Erfurt': ['Frankfurt', 'Wurzburg', 'Erfurt'],
'Numberg': ['Frankfurt', 'Wurzburg', 'Numberg'], 'Stuttgart':
```

3. Minimum Spanning Tree

Now we have another problem. We work for a water pipe laying company or an internet fiber company. ***We need to connect all the cities in the graph we have using the minimum amount of wire/pipe.*** How do we do this?



An Undirected Graph and its MST on the right.

Applications

- Minimum spanning trees have direct applications in the design of networks, including computer networks, telecommunications networks, transportation networks, water supply networks, and electrical grids (which they were first invented for)

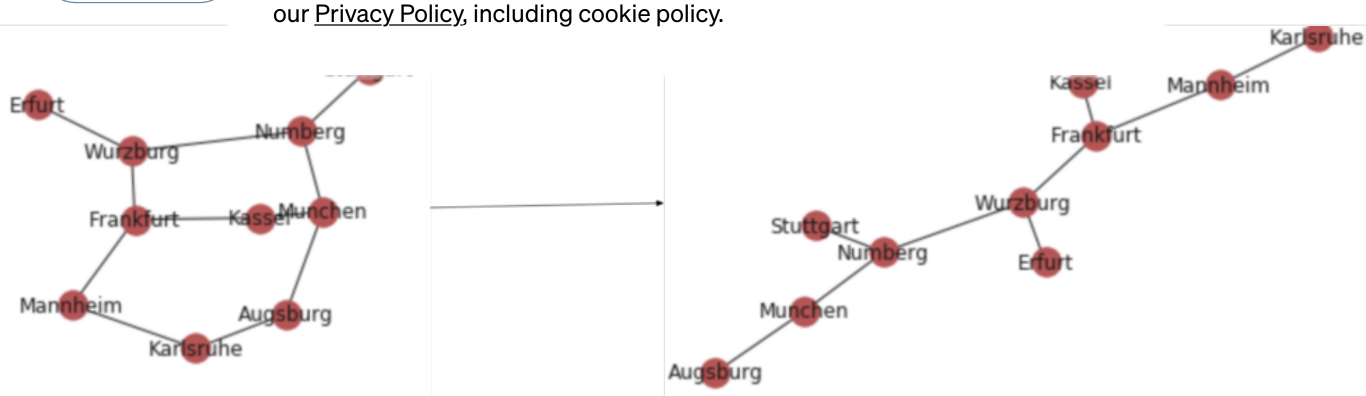
- MST is used for approximating the traveling salesman problem

- Clustering — First construct MST and then determine a threshold value for breaking some edges in the MST using Intercluster distances and Intracluster distances.

- Image Segmentation — It was used for Image segmentation where we first construct an MST on a graph where pixels are nodes and distances between pixels are based on some similarity measure(color, intensity, etc.)

Code

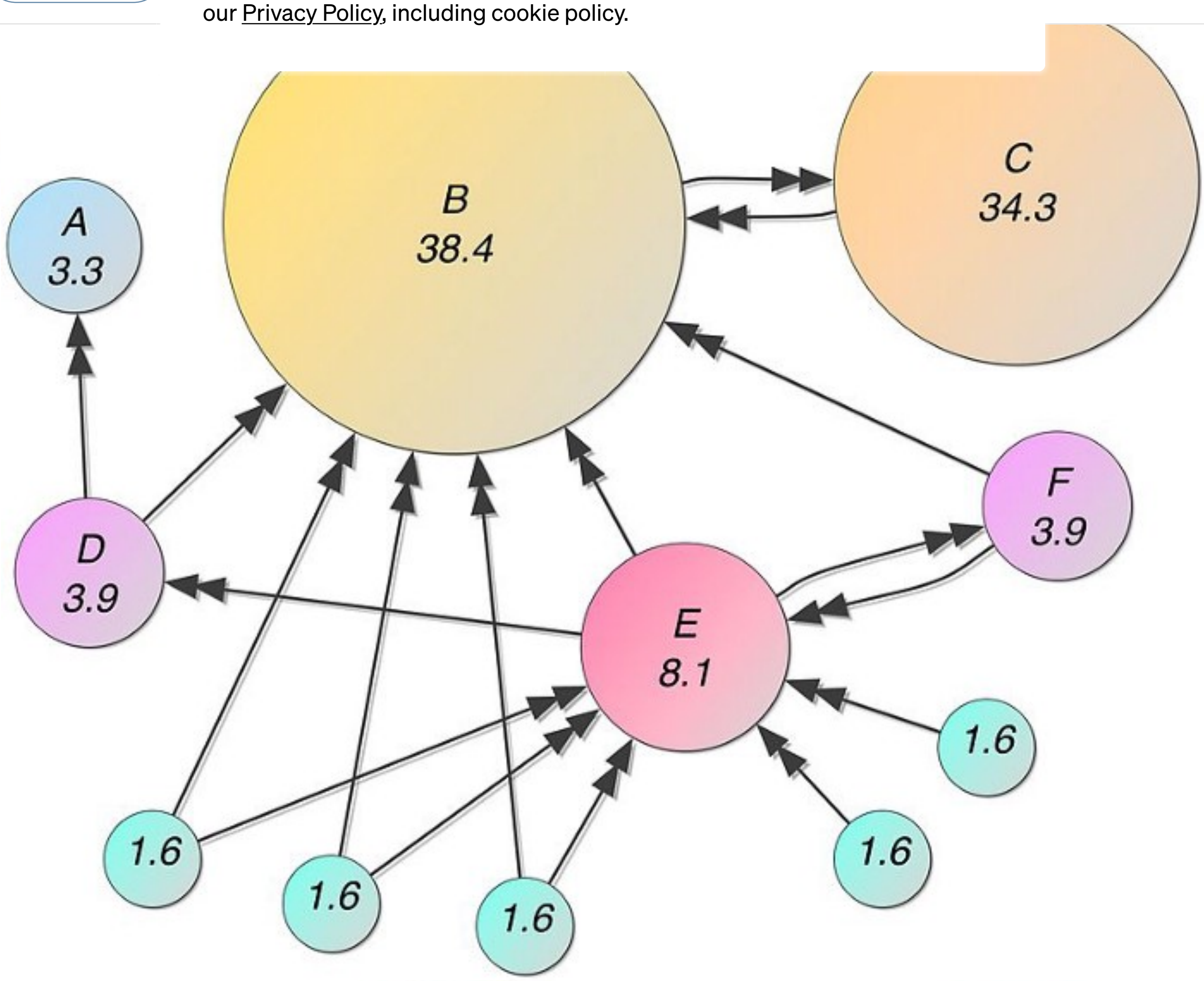
```
# nx.minimum_spanning_tree(g) returns a instance of type graph
nx.draw_networkx(nx.minimum_spanning_tree(g))
```



The MST of our graph.

As you can see the above is the wire we gotta lay.

4. Pagerank



This is the page sorting algorithm that powered google for a long time. It assigns scores to pages based on the number and quality of incoming and outgoing links.

Applications

Pagerank can be used anywhere where we want to estimate node importance in any network.

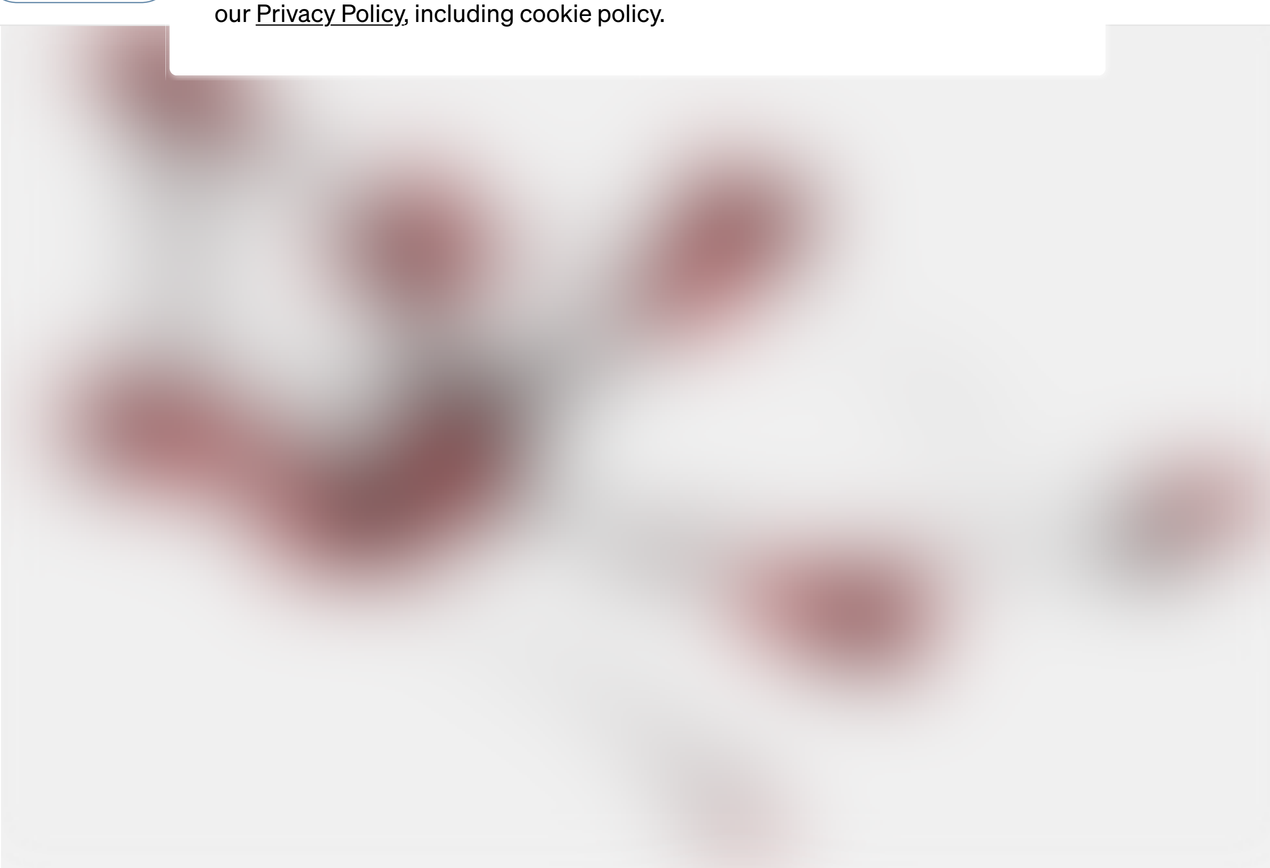
- It has been used for finding the most influential papers using citations.
- Has been used by Google to rank pages
- It can be used to rank tweets- User and Tweets as nodes. Create Link between user if user A follows user B and Link between user and Tweets if user tweets/retweets a tweet.

For this exercise, we are going to be using Facebook data. We have a file of edges/links between facebook users. We first create the FB graph using:

```
# reading the dataset

fb = nx.read_edgelist('../input/facebook-combined.txt', create_using
= nx.Graph(), nodetype = int)
```

This is how it looks:

```
pos = nx.spring_layout(fb)

import warnings
warnings.filterwarnings('ignore')

plt.style.use('fivethirtyeight')
plt.rcParams['figure.figsize'] = (20, 15)
plt.axis('off')
nx.draw_networkx(fb, pos, with_labels = False, node_size = 35)
plt.show()
```



FB User Graph

Now we want to find the users having high influence capability.

Intuitively, the Pagerank algorithm will give a higher score to a user who has a lot of friends who in turn have a lot of FB Friends.

```
pageranks = nx.pagerank(fb)
print(pageranks)
------------------------------------------------------
{0: 0.006289602618466542,
 1: 0.00023590202311540972,
 2: 0.00020310565091694562,
 3: 0.00022552359869430617,
 4: 0.00023849264701222462,
........}
```

We can get the sorted PageRank or most influential users using:

```
import o
sorted_pagerank = sorted(pageranks.items(),
key=operator.itemgetter(1),reverse = True)
print(sorted_pagerank)
------------------------------------------------------
[(3437, 0.007614586844749603), (107, 0.006936420955866114), (1684,
0.0063671621383068295), (0, 0.006289602618466542), (1912,
0.0038769716008844974), (348, 0.0023480969727805783), (686,
0.0022193592598000193), (3980, 0.002170323579009993), (414,
0.0018002990470702262), (698, 0.0013171153138368807), (483,
0.0012974283300616082), (3830, 0.0011844348977671688), (376,
0.0009014073664792464), (2047, 0.000841029154597401), (56,
0.0008039024292749443), (25, 0.000800412660519768), (828,
0.0007886905420662135), (322, 0.0007867992190291396),......]
```

The above IDs are for the most influential users.

We can see the subgraph for the most influential user:

```
first_degree_connected_nodes = list(fb.neighbors(3437))
second_degree_connected_nodes = []
for x in first_degree_connected_nodes:
    second_degree_connected_nodes+=list(fb.neighbors(x))
second_degree_connected_nodes.remove(3437)
second_degree_connected_nodes =
list(set(second_degree_connected_nodes))

subgraph_3437 =
nx.subgraph(fb,first_degree_connected_nodes+second_degree_connected_
nodes)

pos = nx.spring_layout(subgraph_3437)

node_color = ['yellow' if v == 3437 else 'red' for v in
subgraph_3437]
node_size =  [1000 if v == 3437 else 35 for v in subgraph_3437]
plt.style.use('fivethirtyeight')
plt.rcParams['figure.figsize'] = (20, 15)
plt.axis('off')

nx.draw_networkx(subgraph_3437, pos, with_labels = False,
node_color=node_color,node_size=node_size )
plt.show()
```

5. Centrality Measures

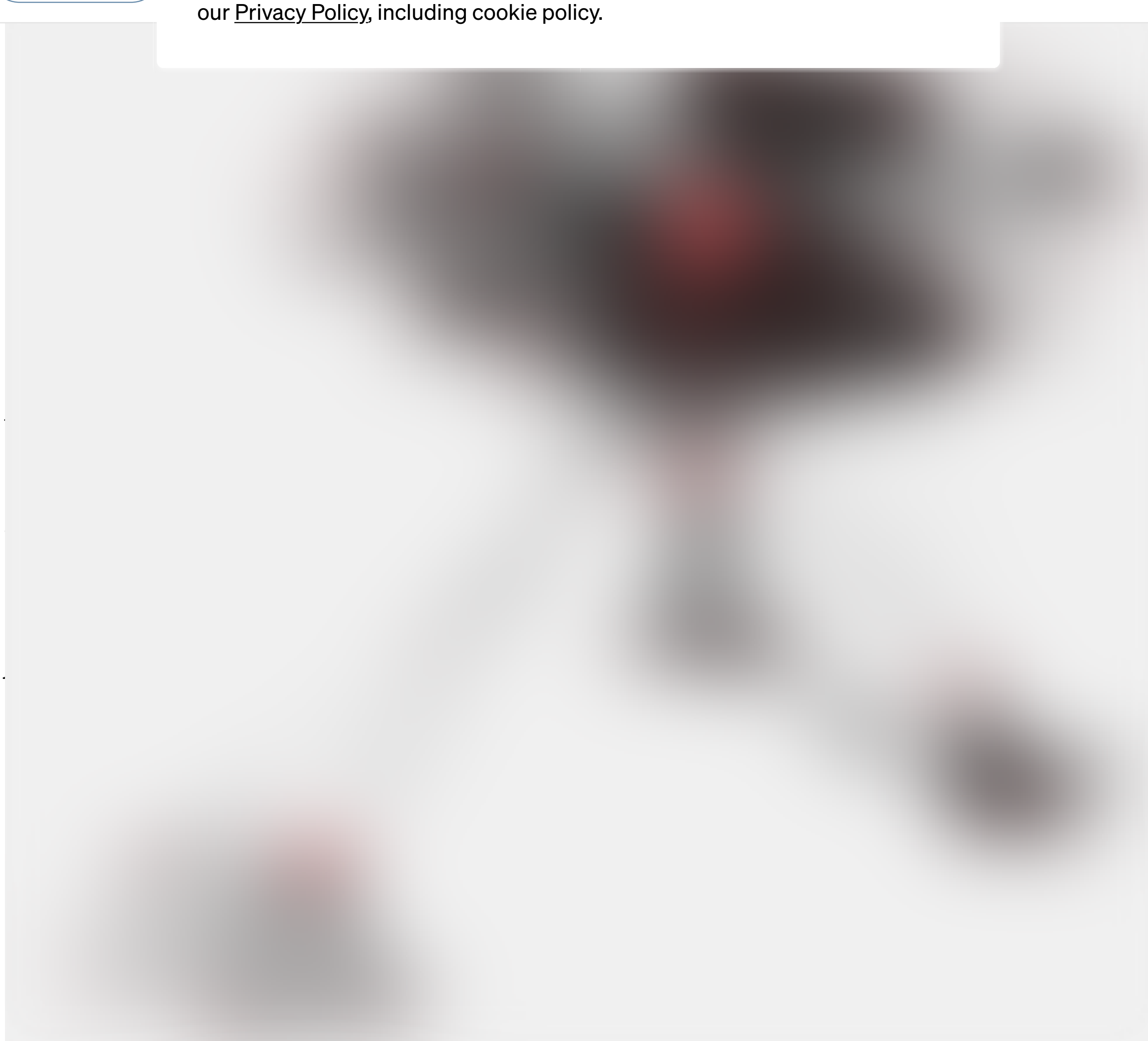
There are a lot of centrality measures which you can use as features to your machine learning models. I will talk about two of them. You can look at other measures here.

Betweenness Centrality: It is not only the users who have the most friends that are important, the users who connect one geography to another are also important as that lets users see content from diverse geographies. ***Betweenness centrality quantifies how many times a particular node comes in the shortest chosen path between two other nodes.***

Degree Centrality: It is simply the number of connections for a node.

Applications

Our most influential user(Yellow)



You can see the nodes sized by their betweenness centrality values here. They can be thought of as information passers. Breaking any of the nodes with a high betweenness Centrality will break the graph into many parts.

If you want to read up more on Graph Algorithms here is a [Graph Analytics for Big Data course on Coursera by UCSanDiego] which I highly recommend to learn the basics of graph theory.

Thanks for the read. I am going to be writing more beginner-friendly posts in the future

@mlwhiz.

Artificial Intelligence Programming Data Science Machine Learning Visualization

assignment4

November 29, 2021

1 Assignment 4

1.1 Description

In this assignment you must read in a file of metropolitan regions and associated sports teams from `assets/wikipedia_data.html` and answer some questions about each metropolitan region. Each of these regions may have one or more teams from the "Big 4": NFL (football, in `assets/nfl.csv`), MLB (baseball, in `assets/mlb.csv`), NBA (basketball, in `assets/nba.csv` or NHL (hockey, in `assets/nhl.csv`). Please keep in mind that all questions are from the perspective of the metropolitan region, and that this file is the "source of authority" for the location of a given sports team. Thus teams which are commonly known by a different area (e.g. "Oakland Raiders") need to be mapped into the metropolitan region given (e.g. San Francisco Bay Area). This will require some human data understanding outside of the data you've been given (e.g. you will have to hand-code some names, and might need to google to find out where teams are)!

For each sport I would like you to answer the question: **what is the win/loss ratio's correlation with the population of the city it is in?** Win/Loss ratio refers to the number of wins over the number of wins plus the number of losses. Remember that to calculate the correlation with `pearsonr`, so you are going to send in two ordered lists of values, the populations from the wikipedia_data.html file and the win/loss ratio for a given sport in the same order. Average the win/loss ratios for those cities which have multiple teams of a single sport. Each sport is worth an equal amount in this assignment (20%*4=80%) of the grade for this assignment. You should only use data **from year 2018** for your analysis – this is important!

1.2 Notes

1. Do not include data about the MLS or CFL in any of the work you are doing, we're only interested in the Big 4 in this assignment.
2. I highly suggest that you first tackle the four correlation questions in order, as they are all similar and worth the majority of grades for this assignment. This is by design!
3. It's fair game to talk with peers about high level strategy as well as the relationship between metropolitan areas and sports teams. However, do not post code solving aspects of the assignment (including such as dictionaries mapping areas to teams, or regexes which will clean up names).
4. There may be more teams than the assert statements test, remember to collapse multiple teams in one city into a single value!

1.3 Question 1

For this question, calculate the win/loss ratio's correlation with the population of the city it is in for the **NHL** using **2018** data.

[38]:
```
import pandas as pd
import numpy as np
import scipy.stats as stats
import re

def nhl_correlation():
    # YOUR CODE HERE
    nhl_df=pd.read_csv("assets/nhl.csv")
    cities=pd.read_html("assets/wikipedia_data.html")[1]
    cities=cities.iloc[:-1,[0,3,5,6,7,8]]
    nhl_df=nhl_df[['team','W','L','year']].drop(0)
    #print(nhl_df[nhl_df['year']==2018])
    nhl_df=nhl_df[nhl_df['year']==2018]
    mask1=nhl_df['team'].isin(['Atlantic Division','Metropolitan␣
↪Division','Central Division','Pacific Division'])
    nhl_df=nhl_df[~mask1]
    pattern="""(?P<team>[\w\s]*)"""
    nhl_df['team']=nhl_df['team'].str.extract(pattern)
    nhl_df['W']=nhl_df['W'].astype('int')
    nhl_df['L']=nhl_df['L'].astype('int')
    pattern2="""(?P<team>[\w\s\-\.]+)"""
    cities['NHL']=cities['NHL'].str.extract(pattern2)

    nhl_df['team']=nhl_df['team'].replace({"Boston Bruins":"Boston",
                                        "Buffalo Sabres":"Buffalo",
                                        "Calgary Flames":"Calgary",
                                        "Columbus Blue Jackets":"Columbus",
                                        "Dallas Stars":"DallasFort Worth",
                                        "Detroit Red Wings":"Detroit",
                                        "Edmonton Oilers":"Edmonton",
                                        "Florida Panthers":"MiamiFort␣
↪Lauderdale",
                                        "Los Angeles Kings":"Los Angeles",
                                        "New York Islanders":"New York City",
                                        "New York Rangers":"New York City",
                                        "Ottawa Senators":"Ottawa",
                                        "Philadelphia Flyers":"Philadelphia",
                                        "Phoenix Coyotes":"Phoenix",
                                        "Pittsburgh Penguins":'Pittsburgh',
                                        "San Jose Sharks":"San Francisco Bay␣
↪Area",
                                        "St":"St. Louis",
                                        "Tampa Bay Lightning":"Tampa Bay Area",
```

```
                                        "Toronto Maple Leafs":"Toronto",
                                        "Vancouver Canucks":"Vancouver",
                                        "Vegas Golden Knights":"Las Vegas",
                                        "Washington Capitals":"Washington, D.C.",
                                        "Winnipeg Jets":"Winnipeg",
                                        "Carolina Hurricanes":"Raleigh",
                                        "Chicago Blackhawks":"Chicago",
                                        "Colorado Avalanche":"Denver",
                                        "Montreal Canadiens":"Montreal",
                                        "Nashville Predators":"Nashville",
                                        "New Jersey Devils":"New York City",
                                            "Arizona Coyotes":'Phoenix',
                                            "Minnesota Wild":"MinneapolisSaint␣
↪Paul"
                                       })
    nhl_df=nhl_df.rename({'team':'city'},axis=1)
    cities=cities.rename({'Metropolitan area':'city','Population (2016 est.
↪)[8]':'population'},axis=1)
    nhl_df["WL"]=nhl_df['W']/(nhl_df['W']+nhl_df['L'])
    nhl_df=nhl_df.groupby("city").agg({"W":np.sum,"L":np.sum,"WL":np.mean})

    result=pd.merge(nhl_df, cities, how='left', on='city')
    result=result.dropna()
    result['population']=result['population'].astype('int64')
    print(result)
    population_by_region = [] # pass in metropolitan area population from␣
↪cities
    win_loss_by_region = [] # pass in win/loss ratio from nhl_df in the same␣
↪order as cities["Metropolitan area"]

    for n in result['population']:
        population_by_region.append(n)
    for n in result['WL']:
        win_loss_by_region.append(n)

    assert len(population_by_region) == len(win_loss_by_region), "Q1: Your␣
↪lists must be the same length"
    assert len(population_by_region) == 28, "Q1: There should be 28 teams being␣
↪analysed for NHL"
    CORR, PVAL=stats.pearsonr(population_by_region, win_loss_by_region)
    return CORR

nhl_correlation()
```

```
                      city   W   L        WL  population  \
1                   Boston  50  20  0.714286     4794447
2                  Buffalo  25  45  0.357143     1132804
```

3	Calgary	37	35	0.513889	1392609
4	Chicago	33	39	0.458333	9512999
5	Columbus	45	30	0.600000	2041520
6	DallasFort Worth	42	32	0.567568	7233323
7	Denver	43	30	0.589041	2853077
8	Detroit	30	39	0.434783	4297617
9	Edmonton	36	40	0.473684	1321426
10	Las Vegas	51	24	0.680000	2155664
11	Los Angeles	45	29	0.608108	13310447
12	MiamiFort Lauderdale	44	30	0.594595	6066387
13	MinneapolisSaint Paul	45	26	0.633803	3551036
14	Montreal	29	40	0.420290	4098927
15	Nashville	53	18	0.746479	1865298
16	New York City	113	105	0.518349	20153634
17	Ottawa	28	43	0.394366	1323783
18	Philadelphia	42	26	0.617647	6070500
19	Phoenix	29	41	0.414286	4661537
20	Pittsburgh	47	29	0.618421	2342299
21	Raleigh	36	35	0.507042	1302946
22	San Francisco Bay Area	45	27	0.625000	6657982
23	St. Louis	44	32	0.578947	2807002
24	Tampa Bay Area	54	23	0.701299	3032171
25	Toronto	49	26	0.653333	5928040
26	Vancouver	31	40	0.436620	2463431
27	Washington, D.C.	49	26	0.653333	6131977
28	Winnipeg	52	20	0.722222	778489

	NFL	MLB	NBA \
1	Patriots[note 14]	Red Sox[note 15]	Celtics
2	Bills[note 56]	[note 57]	[note 58]
3			
4	Bears[note 8]	CubsWhite Sox	Bulls[note 9]
5			
6	Cowboys	Rangers	Mavericks
7	Broncos	Rockies	Nuggets[note 17]
8	Lions	Tigers[note 20]	Pistons[note 21]
9			
10	[note 6]		
11	RamsChargers[note 4]	DodgersAngels	LakersClippers
12	Dolphins	Marlins	Heat
13	Vikings	Twins	Timberwolves
14		[note 59]	
15	Titans		
16	GiantsJets[note 1]	YankeesMets[note 2]	KnicksNets
17			
18	Eagles	Phillies[note 12]	76ers
19	Cardinals	Diamondbacks	Suns
20	Steelers	Pirates	[note 27]

```
21
22  49ersRaiders[note 6]      GiantsAthletics          Warriors
23             [note 40]   Cardinals[note 41]          [note 42]
24            Buccaneers                 Rays
25            [note 22]           Blue Jays  Raptors[note 23]
26                                                 [note 60]
27              Redskins   Nationals[note 10]  Wizards[note 11]
28

                       NHL
1                   Bruins
2                   Sabres
3                   Flames
4               Blackhawks
5             Blue Jackets
6                    Stars
7                Avalanche
8                Red Wings
9                   Oilers
10          Golden Knights
11              KingsDucks
12                Panthers
13                    Wild
14               Canadiens
15               Predators
16  RangersIslandersDevils
17                Senators
18                  Flyers
19                 Coyotes
20                Penguins
21              Hurricanes
22                  Sharks
23                   Blues
24               Lightning
25             Maple Leafs
26                 Canucks
27                Capitals
28                    Jets
```

[38]:
```
0.002279772528901799
```

[ ]:
```

```

1.4 Question 2

For this question, calculate the win/loss ratio's correlation with the population of the city it is in for the **NBA** using **2018** data.

[53]:
```
import pandas as pd
import numpy as np
import scipy.stats as stats
import re


def nba_correlation():

    nba_df=pd.read_csv("assets/nba.csv")
    cities=pd.read_html("assets/wikipedia_data.html")[1]
    cities=cities.iloc[:-1,[0,3,5,6,7,8]]
    nba_df=nba_df[['team','W','L','year']]
    nba_df=nba_df[nba_df['year']==2018]
    pattern="""(?P<team>[\w\s]*)"""
    nba_df['team']=nba_df['team'].str.extract(pattern)
    nba_df['team']=nba_df['team'].str.strip()
    nba_df['W']=nba_df['W'].astype('int')
    nba_df['L']=nba_df['L'].astype('int')

    nba_df['team']=nba_df['team'].replace({"Toronto Raptors":"Toronto",
                                            "Boston Celtics":"Boston",
                                            "Philadelphia 76ers":"Philadelphia",
                                            "Cleveland Cavaliers":"Cleveland",
                                            "Indiana Pacers":"Indianapolis",
                                            "Miami Heat":"MiamiFort Lauderdale",
                                            "Milwaukee Bucks":"Milwaukee",
                                            "Washington Wizards":"Washington, D.
 ↪C.",
                                            "Detroit Pistons":"Detroit",
                                            "Charlotte Hornets":"Charlotte",
                                            "New York Knicks":"New York City",
                                            "Brooklyn Nets":"New York City",
                                            "Chicago Bulls":"Chicago",
                                            "Orlando Magic":"Orlando",
                                            "Atlanta Hawks":"Atlanta",
                                            "Houston Rockets":"Houston",
                                            "Golden State Warriors":"San␣
 ↪Francisco Bay Area",
                                            "Portland Trail Blazers":"Portland",
                                            "Oklahoma City Thunder":"Oklahoma␣
 ↪City",
                                            "Utah Jazz":"Salt Lake City",
                                            "New Orleans Pelicans":"New Orleans",
                                            "San Antonio Spurs":"San Antonio",
                                            "Minnesota Timberwolves":
 ↪"MinneapolisSaint Paul",
                                            "Denver Nuggets":"Denver",
```

```
                                            "Los Angeles Clippers":"Los Angeles",
                                            "Sacramento Kings":"Sacramento",
                                            "Dallas Mavericks":"DallasFort␣
↪Worth",
                                            "Memphis Grizzlies":"Memphis",
                                            "Phoenix Suns":"Phoenix"
                                       })
    nba_df=nba_df.rename({'team':'city'},axis=1)
    cities=cities.rename({'Metropolitan area':'city','Population (2016 est.
↪)[8]':'population'},axis=1)
    nba_df["WL"]=nba_df['W']/(nba_df['W']+nba_df['L'])
    nba_df=nba_df.groupby("city").agg({"W":np.sum,"L":np.sum,"WL":np.mean})

    result=pd.merge(nba_df, cities, how='left', on='city')
    result=result.dropna()
    result['population']=result['population'].astype('int64')
    print(result)
    population_by_region = [] # pass in metropolitan area population from␣
↪cities
    win_loss_by_region = [] # pass in win/loss ratio from nhl_df in the same␣
↪order as cities["Metropolitan area"]

    for n in result['population']:
        population_by_region.append(n)
    for n in result['WL']:
        win_loss_by_region.append(n)

# pass in win/loss ratio from nba_df in the same order as cities["Metropolitan␣
↪area"]
    CORR, PVAL=stats.pearsonr(population_by_region, win_loss_by_region)
    assert len(population_by_region) == len(win_loss_by_region), "Q2: Your␣
↪lists must be the same length"
    assert len(population_by_region) == 28, "Q2: There should be 28 teams being␣
↪analysed for NBA"

    return CORR
nba_correlation()
```

```
                      city   W   L        WL  population  \
0                  Atlanta  24  58  0.292683     5789700
1                   Boston  55  27  0.670732     4794447
2                Charlotte  36  46  0.439024     2474314
3                  Chicago  27  55  0.329268     9512999
4                Cleveland  50  32  0.609756     2055612
5        DallasFort Worth  24  58  0.292683     7233323
6                   Denver  46  36  0.560976     2853077
7                  Detroit  39  43  0.475610     4297617
```

8	Houston	65	17	0.792683	6772470
9	Indianapolis	48	34	0.585366	2004230
10	Los Angeles	42	40	0.512195	13310447
12	Memphis	22	60	0.268293	1342842
13	MiamiFort Lauderdale	44	38	0.536585	6066387
14	Milwaukee	44	38	0.536585	1572482
15	MinneapolisSaint Paul	47	35	0.573171	3551036
16	New Orleans	48	34	0.585366	1268883
17	New York City	57	107	0.347561	20153634
18	Oklahoma City	48	34	0.585366	1373211
19	Orlando	25	57	0.304878	2441257
20	Philadelphia	52	30	0.634146	6070500
21	Phoenix	21	61	0.256098	4661537
22	Portland	49	33	0.597561	2424955
23	Sacramento	27	55	0.329268	2296418
24	Salt Lake City	48	34	0.585366	1186187
25	San Antonio	47	35	0.573171	2429609
26	San Francisco Bay Area	58	24	0.707317	6657982
27	Toronto	59	23	0.719512	5928040
28	Washington, D.C.	43	39	0.524390	6131977

	NFL	MLB	NBA	\
0	Falcons	Braves	Hawks	
1	Patriots[note 14]	Red Sox[note 15]	Celtics	
2	Panthers		Hornets[note 49]	
3	Bears[note 8]	CubsWhite Sox	Bulls[note 9]	
4	Browns[note 29]	Indians[note 30]	Cavaliers[note 31]	
5	Cowboys	Rangers	Mavericks	
6	Broncos	Rockies	Nuggets[note 17]	
7	Lions	Tigers[note 20]	Pistons[note 21]	
8	Texans[note 24]	Astros	Rockets	
9	Colts	[note 50]	Pacers[note 51]	
10	RamsChargers[note 4]	DodgersAngels	LakersClippers	
12	[note 69]		Grizzlies	
13	Dolphins	Marlins	Heat	
14	[note 53]	Brewers[note 54]	Bucks	
15	Vikings	Twins	Timberwolves	
16	Saints		Pelicans[note 55]	
17	GiantsJets[note 1]	YankeesMets[note 2]	KnicksNets	
18			Thunder[note 68]	
19			Magic	
20	Eagles	Phillies[note 12]	76ers	
21	Cardinals	Diamondbacks	Suns	
22			Trail Blazers	
23			Kings	
24			Jazz	
25	[note 64]		Spurs	
26	49ersRaiders[note 6]	GiantsAthletics	Warriors	

```
27             [note 22]             Blue Jays     Raptors[note 23]
28              Redskins    Nationals[note 10]     Wizards[note 11]

                                 NHL
0                          [note 25]
1                             Bruins
2
3                         Blackhawks
4                          [note 32]
5                              Stars
6               Avalanche[note 18]
7                          Red Wings
8
9
10                        KingsDucks
12
13                          Panthers
14
15                    Wild[note 16]
16
17   RangersIslandersDevils[note 3]
18
19
20                  Flyers[note 13]
21                           Coyotes
22
23
24
25
26                  Sharks[note 7]
27                       Maple Leafs
28                          Capitals
```

[53]:
```
-0.15535519103613454
```

[ ]:
```
```

1.5 Question 3

For this question, calculate the win/loss ratio's correlation with the population of the city it is in for the **MLB** using **2018** data.

[61]:
```python
import pandas as pd
import numpy as np
import scipy.stats as stats
import re
```

```python
def mlb_correlation():
    mlb_df=pd.read_csv("assets/mlb.csv")
    cities=pd.read_html("assets/wikipedia_data.html")[1]
    cities=cities.iloc[:-1,[0,3,5,6,7,8]]

    mlb_df=mlb_df[['team','W','L','year']]
    mlb_df=mlb_df[mlb_df['year']==2018]
    mlb_df['team']=mlb_df['team'].str.strip()
    mlb_df['W']=mlb_df['W'].astype('int')
    mlb_df['L']=mlb_df['L'].astype('int')

    mlb_df['team']=mlb_df['team'].replace({"Boston Red Sox":"Boston",
                                           "New York Yankees":"New York City",
                                           "Tampa Bay Rays":"Tampa Bay Area",
                                           "Toronto Blue Jays":"Toronto",
                                           "Baltimore Orioles":"Baltimore",
                                           "Cleveland Indians":"Cleveland",
                                           "Minnesota Twins":"MinneapolisSaint␣
 ↪Paul",
                                           "Detroit Tigers":"Detroit",
                                           "Chicago White Sox":"Chicago",
                                           "Kansas City Royals":"Kansas City",
                                           "Houston Astros":"Houston",
                                           "Oakland Athletics":"San Francisco␣
 ↪Bay Area",
                                           "Seattle Mariners":"Seattle",
                                           "Los Angeles Angels":"Los Angeles",
                                           "Texas Rangers":"DallasFort Worth",
                                           "Atlanta Braves":"Atlanta",
                                           "Washington Nationals":"Washington,␣
 ↪D.C.",
                                           "Philadelphia Phillies":
 ↪"Philadelphia",
                                           "New York Mets":"New York City",
                                           "Miami Marlins":"MiamiFort␣
 ↪Lauderdale",
                                           "Milwaukee Brewers":"Milwaukee",
                                           "Chicago Cubs":"Chicago",
                                           "St. Louis Cardinals":"St. Louis",
                                           "Pittsburgh Pirates":"Pittsburgh",
                                           "Cincinnati Reds":"Cincinnati",
                                           "Los Angeles Dodgers":"Los Angeles",
                                           "Colorado Rockies":"Denver",
                                           "Arizona Diamondbacks":"Phoenix",
                                           "San Francisco Giants":"San␣
 ↪Francisco Bay Area",
```

```
                                                    "San Diego Padres":"San Diego"
                                              })
    mlb_df=mlb_df.rename({'team':'city'},axis=1)
    cities=cities.rename({'Metropolitan area':'city','Population (2016 est.
↪)[8]':'population'},axis=1)
    mlb_df["WL"]=mlb_df['W']/(mlb_df['W']+mlb_df['L'])
    mlb_df=mlb_df.groupby("city").agg({"W":np.sum,"L":np.sum,"WL":np.mean})

    result=pd.merge(mlb_df, cities, how='left', on='city')
    result=result.dropna()
    result['population']=result['population'].astype('int64')
    print(result)
    population_by_region = [] # pass in metropolitan area population from␣
↪cities
    win_loss_by_region = [] # pass in win/loss ratio from nhl_df in the same␣
↪order as cities["Metropolitan area"]

    for n in result['population']:
        population_by_region.append(n)
    for n in result['WL']:
        win_loss_by_region.append(n)

    assert len(population_by_region) == len(win_loss_by_region), "Q3: Your␣
↪lists must be the same length"
    assert len(population_by_region) == 26, "Q3: There should be 26 teams being␣
↪analysed for MLB"
    corr,pval=stats.pearsonr(population_by_region, win_loss_by_region)
    return corr
mlb_correlation()
```

```
                       city    W    L        WL  population  \
0                   Atlanta   90   72  0.555556     5789700
1                 Baltimore   47  115  0.290123     2798886
2                    Boston  108   54  0.666667     4794447
3                   Chicago  157  168  0.483077     9512999
4                Cincinnati   67   95  0.413580     2165139
5                 Cleveland   91   71  0.561728     2055612
6          DallasFort Worth   67   95  0.413580     7233323
7                    Denver   91   72  0.558282     2853077
8                   Detroit   64   98  0.395062     4297617
9                   Houston  103   59  0.635802     6772470
10              Kansas City   58  104  0.358025     2104509
11              Los Angeles  172  153  0.529231    13310447
12     MiamiFort Lauderdale   63   98  0.391304     6066387
13                Milwaukee   96   67  0.588957     1572482
14   MinneapolisSaint Paul   78   84  0.481481     3551036
15            New York City  177  147  0.546296    20153634
```

17	Philadelphia	80	82	0.493827	6070500
18	Phoenix	82	80	0.506173	4661537
19	Pittsburgh	82	79	0.509317	2342299
20	San Diego	66	96	0.407407	3317749
21	San Francisco Bay Area	73	89	0.450617	6657982
22	Seattle	89	73	0.549383	3798902
23	St. Louis	88	74	0.543210	2807002
24	Tampa Bay Area	90	72	0.555556	3032171
25	Toronto	73	89	0.450617	5928040
26	Washington, D.C.	82	80	0.506173	6131977

	NFL	MLB	NBA \
0	Falcons	Braves	Hawks
1	Ravens[note 45]	Orioles[note 46]	[note 47]
2	Patriots[note 14]	Red Sox[note 15]	Celtics
3	Bears[note 8]	CubsWhite Sox	Bulls[note 9]
4	Bengals	Reds[note 35]	[note 36]
5	Browns[note 29]	Indians[note 30]	Cavaliers[note 31]
6	Cowboys	Rangers	Mavericks
7	Broncos	Rockies	Nuggets[note 17]
8	Lions	Tigers[note 20]	Pistons[note 21]
9	Texans[note 24]	Astros	Rockets
10	Chiefs	Royals[note 37]	[note 38]
11	RamsChargers[note 4]	DodgersAngels	LakersClippers
12	Dolphins	Marlins	Heat
13	[note 53]	Brewers[note 54]	Bucks
14	Vikings	Twins	Timberwolves
15	GiantsJets[note 1]	YankeesMets[note 2]	KnicksNets
17	Eagles	Phillies[note 12]	76ers
18	Cardinals	Diamondbacks	Suns
19	Steelers	Pirates	[note 27]
20	[note 62]	Padres	[note 63]
21	49ersRaiders[note 6]	GiantsAthletics	Warriors
22	Seahawks	Mariners	[note 33]
23	[note 40]	Cardinals[note 41]	[note 42]
24	Buccaneers	Rays	
25	[note 22]	Blue Jays	Raptors[note 23]
26	Redskins	Nationals[note 10]	Wizards[note 11]

	NHL
0	[note 25]
1	
2	Bruins
3	Blackhawks
4	
5	[note 32]
6	Stars
7	Avalanche[note 18]

```
8                               Red Wings
9
10                              [note 39]
11                             KingsDucks
12                               Panthers
13
14                         Wild[note 16]
15   RangersIslandersDevils[note 3]
17                       Flyers[note 13]
18                                Coyotes
19                     Penguins[note 28]
20
21                        Sharks[note 7]
22                              [note 34]
23                        Blues[note 43]
24                              Lightning
25                            Maple Leafs
26                               Capitals
```

[61]:
```
0.13918951993280002
```

[ ]:

1.6 Question 4

For this question, calculate the win/loss ratio's correlation with the population of the city it is in for the **NFL** using **2018** data.

[3]:
```python
import pandas as pd
import numpy as np
import scipy.stats as stats
import re



def nfl_correlation():
    nfl_df=pd.read_csv("assets/nfl.csv")
    cities=pd.read_html("assets/wikipedia_data.html")[1]
    cities=cities.iloc[:-1,[0,3,5,6,7,8]]
    nfl_df=nfl_df[['team','W','L','year']].drop(0)
    nfl_df=nfl_df[nfl_df['year']==2018]
    #print(nfl_df)
    mask1=nfl_df['team'].isin(['AFC North','AFC South','AFC West','NFC␣
↪East','NFC North','NFC South','NFC West'])
    nfl_df=nfl_df[~mask1]

    pattern="""(?P<team>[\w\s]*)"""
    nfl_df['team']=nfl_df['team'].str.extract(pattern)
```

```
    nfl_df['W']=nfl_df['W'].astype('int')
    nfl_df['L']=nfl_df['L'].astype('int')
    nfl_df['team']=nfl_df['team'].replace({"New England Patriots":"Boston",
                                          "Miami Dolphins":"MiamiFort␣
↪Lauderdale",
                                          "Buffalo Bills":"Buffalo",
                                          "New York Jets":"New York City",
                                          "Baltimore Ravens":"Baltimore",
                                          "Pittsburgh Steelers":"Pittsburgh",
                                          "Cleveland Browns":"Cleveland",
                                          "Cincinnati Bengals":"Cincinnati",
                                          "Houston Texans":"Houston",
                                          "Indianapolis Colts":"Indianapolis",
                                          "Tennessee Titans":"Nashville",
                                          "Jacksonville Jaguars":
↪"Jacksonville",
                                          "Kansas City Chiefs":"Kansas City",
                                          "Los Angeles Chargers":"Los Angeles",
                                          "Denver Broncos":"Denver",
                                          "Oakland Raiders":"San Francisco Bay␣
↪Area",
                                          "Dallas Cowboys":"DallasFort Worth",
                                          "Philadelphia Eagles":"Philadelphia",
                                          "Washington Redskins":"Washington, D.
↪C.",
                                          "New York Giants":"New York City",
                                          "Chicago Bears":"Chicago",
                                          "Minnesota Vikings":
↪"MinneapolisSaint Paul",
                                          "Green Bay Packers":"Green Bay",
                                          "Detroit Lions":"Detroit",
                                          "New Orleans Saints":"New Orleans",
                                          "Carolina Panthers":"Charlotte",
                                          "Atlanta Falcons":"Atlanta",
                                          "Tampa Bay Buccaneers":"Tampa Bay␣
↪Area",
                                          "Los Angeles Rams":"Los Angeles",
                                          "Seattle Seahawks":"Seattle",
                                          "San Francisco 49ers":"San Francisco␣
↪Bay Area",
                                          "Arizona Cardinals":"Phoenix"
                                     })
    nfl_df=nfl_df.rename({'team':'city'},axis=1)
    cities=cities.rename({'Metropolitan area':'city','Population (2016 est.
↪)[8]':'population'},axis=1)
    nfl_df["WL"]=nfl_df['W']/(nfl_df['W']+nfl_df['L'])
```

```
    nfl_df=nfl_df.groupby("city").agg({"W":np.sum,"L":np.sum,"WL":np.mean})

    result=pd.merge(nfl_df, cities, how='left', on='city')
    #result=result.dropna()
    result['population']=result['population'].astype('int64')
    print(result)
    population_by_region = [] # pass in metropolitan area population from␣
↪cities
    win_loss_by_region = [] # pass in win/loss ratio from nhl_df in the same␣
↪order as cities["Metropolitan area"]

    for n in result['population']:
        population_by_region.append(n)
    for n in result['WL']:
        win_loss_by_region.append(n)


    assert len(population_by_region) == len(win_loss_by_region), "Q4: Your␣
↪lists must be the same length"
    assert len(population_by_region) == 29, "Q4: There should be 29 teams being␣
↪analysed for NFL"
    CORR, PVAL=stats.pearsonr(population_by_region, win_loss_by_region)
    return CORR

nfl_correlation()
```

```
                     city   W   L        WL  population  \
0                 Atlanta   7   9  0.437500     5789700
1               Baltimore  10   6  0.625000     2798886
2                  Boston  11   5  0.687500     4794447
3                 Buffalo   6  10  0.375000     1132804
4               Charlotte   7   9  0.437500     2474314
5                 Chicago  12   4  0.750000     9512999
6              Cincinnati   6  10  0.375000     2165139
7               Cleveland   7   8  0.466667     2055612
8        DallasFort Worth  10   6  0.625000     7233323
9                  Denver   6  10  0.375000     2853077
10                Detroit   6  10  0.375000     4297617
11              Green Bay   6   9  0.400000      318236
12                Houston  11   5  0.687500     6772470
13           Indianapolis  10   6  0.625000     2004230
14           Jacksonville   5  11  0.312500     1478212
15            Kansas City  12   4  0.750000     2104509
16            Los Angeles  25   7  0.781250    13310447
17    MiamiFort Lauderdale   7   9  0.437500     6066387
18   MinneapolisSaint Paul   8   7  0.533333     3551036
19              Nashville   9   7  0.562500     1865298
```

20	New Orleans	13	3	0.812500	1268883
21	New York City	9	23	0.281250	20153634
22	Philadelphia	9	7	0.562500	6070500
23	Phoenix	3	13	0.187500	4661537
24	Pittsburgh	9	6	0.600000	2342299
25	San Francisco Bay Area	8	24	0.250000	6657982
26	Seattle	10	6	0.625000	3798902
27	Tampa Bay Area	5	11	0.312500	3032171
28	Washington, D.C.	7	9	0.437500	6131977

	NFL	MLB	NBA \
0	Falcons	Braves	Hawks
1	Ravens[note 45]	Orioles[note 46]	[note 47]
2	Patriots[note 14]	Red Sox[note 15]	Celtics
3	Bills[note 56]	[note 57]	[note 58]
4	Panthers		Hornets[note 49]
5	Bears[note 8]	CubsWhite Sox	Bulls[note 9]
6	Bengals	Reds[note 35]	[note 36]
7	Browns[note 29]	Indians[note 30]	Cavaliers[note 31]
8	Cowboys	Rangers	Mavericks
9	Broncos	Rockies	Nuggets[note 17]
10	Lions	Tigers[note 20]	Pistons[note 21]
11	Packers		
12	Texans[note 24]	Astros	Rockets
13	Colts	[note 50]	Pacers[note 51]
14	Jaguars		
15	Chiefs	Royals[note 37]	[note 38]
16	RamsChargers[note 4]	DodgersAngels	LakersClippers
17	Dolphins	Marlins	Heat
18	Vikings	Twins	Timberwolves
19	Titans		
20	Saints		Pelicans[note 55]
21	GiantsJets[note 1]	YankeesMets[note 2]	KnicksNets
22	Eagles	Phillies[note 12]	76ers
23	Cardinals	Diamondbacks	Suns
24	Steelers	Pirates	[note 27]
25	49ersRaiders[note 6]	GiantsAthletics	Warriors
26	Seahawks	Mariners	[note 33]
27	Buccaneers	Rays	
28	Redskins	Nationals[note 10]	Wizards[note 11]

	NHL
0	[note 25]
1	
2	Bruins
3	Sabres
4	
5	Blackhawks

```
6
7                         [note 32]
8                             Stars
9                Avalanche[note 18]
10                        Red Wings
11
12
13
14
15                        [note 39]
16                      KingsDucks
17                         Panthers
18                    Wild[note 16]
19                        Predators
20
21  RangersIslandersDevils[note 3]
22                  Flyers[note 13]
23                          Coyotes
24                Penguins[note 28]
25                   Sharks[note 7]
26                        [note 34]
27                        Lightning
28                         Capitals
```

[3]: 0.004922112149349393

[ ]:

1.7 Question 5

In this question I would like you to explore the hypothesis that **given that an area has two sports teams in different sports, those teams will perform the same within their respective sports**. How I would like to see this explored is with a series of paired t-tests (so use `ttest_rel`) between all pairs of sports. Are there any sports where we can reject the null hypothesis? Again, average values where a sport has multiple teams in one region. Remember, you will only be including, for each sport, cities which have teams engaged in that sport, drop others as appropriate. This question is worth 20% of the grade for this assignment.

[ ]:
```python
import pandas as pd
import numpy as np
import scipy.stats as stats
import re

mlb_df=pd.read_csv("assets/mlb.csv")
nhl_df=pd.read_csv("assets/nhl.csv")
nba_df=pd.read_csv("assets/nba.csv")
nfl_df=pd.read_csv("assets/nfl.csv")
cities=pd.read_html("assets/wikipedia_data.html")[1]
```

```python
cities=cities.iloc[:-1,[0,3,5,6,7,8]]

def sports_team_performance():
    # YOUR CODE HERE
    raise NotImplementedError()

    # Note: p_values is a full dataframe, so df.loc["NFL","NBA"] should be the same as df.loc["NBA","NFL"] and
    # df.loc["NFL","NFL"] should return np.nan
    sports = ['NFL', 'NBA', 'NHL', 'MLB']
    p_values = pd.DataFrame({k:np.nan for k in sports}, index=sports)

    assert abs(p_values.loc["NBA", "NHL"] - 0.02) <= 1e-2, "The NBA-NHL p-value should be around 0.02"
    assert abs(p_values.loc["MLB", "NFL"] - 0.80) <= 1e-2, "The MLB-NFL p-value should be around 0.80"
    return p_values
```

[ ]:

```python

```