KEY_Lesson16_Basics_Stats_III

July 12, 2019

1 Introduction to Statistics Part III

Now that we have learned how to manipulate basic statistics, we will look at how to perform *significance tests* and find the *correlation* between two variables.

1.1 Significance Testing

[8]:
```
# Load pandas, numpy, and scipy.stats
import pandas as pd
import numpy as np
import scipy.stats as stats
# mount Google Drive
from google.colab import drive
drive.mount('/content/gdrive')
path = '/content/gdrive/My Drive/SummerExperience-master/'
```

Run the cell below to load a table of temperature in Detroit since 1937:

[9]:
```
data_table = pd.read_csv(path + 'Lessons/SampleData/detroit_weather_2.csv' ) #␣
↪Data from Mathematica WeatherData, 2019
```

Print out the data to see what the format looks like:

[10]:
```
# View the head of data_table to see what its format looks like
data_table.head()
```

[10]:

	YEAR	MONTH	DAY	Temperature
0	1937	1	1	0.50
1	1937	1	2	0.17
2	1937	1	3	-1.06
3	1937	1	4	-3.89
4	1937	1	5	-0.17

[11]:
```
# View the tail of data_table to see what its format looks like
data_table.tail()
```

[11]:

	YEAR	MONTH	DAY	Temperature
30057	2019	4	27	10.00
30058	2019	4	28	4.89
30059	2019	4	29	5.67
30060	2019	4	30	6.44

```
30061  2019      5    1        11.22
```

`data_table` contains one row for each day since 1937, where the column 'Temperature' contains the average temperature for that day (in Celsius).

We will use this data to **test** whether or not the average temperature in Detroit has changed significantly in the years since 1937. To do this, let's first select two equally sized date ranges to generate our averages: 1940-1950 and 2005-2015.

[12]:
```
# Select two temperature ranges from data_table, one from a long time ago and␣
 ↪one more recent:

temps_1940 = data_table.query("YEAR >= 1940 and YEAR < 1950")["Temperature"]
temps_2005 = data_table.query("YEAR >= 2005 and YEAR < 2015")["Temperature"]
```

Using what we learned this morning, calculate the mean for each of the date ranges.

[13]:
```
# Calculate the mean of your two temperature ranges:

print("Average temperature 1935-1945:", np.mean(temps_1940))
print("Average temperature 2005-2015:", np.mean(temps_2005))
```

```
Average temperature 1935-1945: 9.48353134410074
Average temperature 2005-2015: 10.230466136550573
```

[14]:
```
# Calculate the difference between the two means:

print("Difference between the means:", np.mean(temps_2005) - np.
 ↪mean(temps_1940))
```

```
Difference between the means: 0.7469347924498333
```

Here, we see that there was an increase of 0.75 degrees Celsius between the average temperature of these two time periods. *Statistical tests* are used to determine if this difference is likely due to chance or due to an actual change.

We will use one of these tests, a **t-test** to calculate the **probability** of this temperature change.

For this, we'll use the `ttest_ind` **function** as part of the `stats` **module** of the `scipy` **package**. You'll notice that the arguments we passed to `ttest_ind` are the full daily temp data vectors for each date range, rather than just the averages. This is because the outcome of the test depends on the **distribution** of the data.

[16]:
```
# Use the scipy stats module to calculate a t-test from the data above

stats.ttest_ind(temps_1940, temps_2005).pvalue
```

[16]:
```
0.0025808160556977724
```

The output of the t-test is called a *p-value*. This *p-value* tells us the **probability** that we would see the same data distribution if there was no difference between the two groups we are testing. Here, the result informs us that there is only a 0.25% chance that there was a difference of this size by random fluctuation, which is very low! Since we saw the average of the later dates was higher than the earlier dates, this shows that our data supports the idea of global warming, even here in Detroit.

Let's redo this analysis using only temperature values from December:

[17]:
```
# Reselect the data, now only including data points in December

temps_1940_dec = data_table.query("YEAR >= 1940 and YEAR < 1950 and MONTH ==␣
 ↪12")["Temperature"]
temps_2005_dec = data_table.query("YEAR >= 2005 and YEAR < 2015 and MONTH ==␣
 ↪12")["Temperature"]
```

[18]:
```
# Calculate the mean of your two temperature ranges:

print("Average temperature 1935-1945:", np.mean(temps_1940_dec))
print("Average temperature 2005-2015:", np.mean(temps_2005_dec))
```

```
Average temperature 1935-1945: -1.7202903225806458
Average temperature 2005-2015: -0.41761437908496746
```

[19]:
```
# Calculate the difference between the two means:

print("Difference between the means:", np.mean(temps_2005_dec) - np.
 ↪mean(temps_1940_dec))
```

```
Difference between the means: 1.3026759434956783
```

[20]:
```
# Re-run the statistical test on these subset datasets

stats.ttest_ind(temps_1940_dec, temps_2005_dec).pvalue
```

[20]:
```
0.0008126329856967648
```

We can see that the difference in temperature is even greater when you focus on just December. A *p-value* of 0.08% indicates that the change is even more signifcant than the difference in temperature for the entire year.

1.2 Correlations

A *correlation* is a measure of the statistical relationship between two variables. Correlation values range from -1 to 1, where the absolute value of the correlation indicates the strength of the relationship and the sign of the correlation represents the direction of the relationship.

We will use the `corrcoef` function from `numpy` to calculate correlation values.

[41]:
```
# positive and negative correlation examples
data_1 = np.array([1,2,3,4,5])
data_2 = data_1 * 4
data_3 = data_1 * -2

print(np.corrcoef(data_1, data_2))
print(np.corrcoef(data_1, data_3))
```

```
[[1. 1.]
 [1. 1.]]
[[ 1. -1.]
 [-1.  1.]]
```

This function returns a *correlation matrix*, which always has 1's along the diagonal and is *symmetric* (i.e. same values above the diagonal as below). This is so you can compute correlations of more than one variable. Let's illustrate with another example.

[42]:
```
a = np.array([1,2,3,4,6,7,8,9])
b = np.array([2,4,6,8,10,12,13,15])
c = np.array([-1,-2,-2,-3,-4,-6,-7,-8])
print(np.corrcoef([a,b,c]))
```

```
[[ 1.          0.99535001 -0.9805214 ]
 [ 0.99535001  1.         -0.97172394]
 [-0.9805214  -0.97172394  1.        ]]
```

The resulting correlation matrix follows the following form:

_	a	b	c
a	1	0.995	-0.980
b	0.995	1	-0.971
c	-0.980	-0.971	1

Now, it should be clear why a correlation matrix always has 1's along the diagonal - every variable has perfect positive correlation with itself. Furthermore, it is symmetric because the correlation of `a` & `b` is the same as the correlation of `b` & `a`.

Now that we understand our output, let's check the correlations between the variables in the `iris` dataset.

[43]:
```
# load and preview iris
iris = pd.read_csv('../SampleData/iris.csv')
iris.head(10)
```

[43]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa
5	5.4	3.9	1.7	0.4	setosa
6	4.6	3.4	1.4	0.3	setosa
7	5.0	3.4	1.5	0.2	setosa
8	4.4	2.9	1.4	0.2	setosa
9	4.9	3.1	1.5	0.1	setosa

[44]:
```
# find correlations between sepal_length, sepal_width, petal_length,␣
 ↪petal_width
```

```
np.corrcoef(iris.iloc[:,0:4], rowvar=False)
```

[44]:
```
array([[ 1.        , -0.11756978,  0.87175378,  0.81794113],
       [-0.11756978,  1.        , -0.4284401 , -0.36612593],
       [ 0.87175378, -0.4284401 ,  1.        ,  0.96286543],
       [ 0.81794113, -0.36612593,  0.96286543,  1.        ]])
```

You'll notice this time we included the `rowvar` parameter - this is because, by default, the `corrcoef` function expects that each row represents a variable, with observations in the columns. In our case it is the opposite - each column represents a variable, while the rows contain observations. So here we change the value of `rowvar` from the default `True` to `False`.

In this lesson you learned how to:

- Perform a `t-test` on a two-class dataset
- Interpret the results from statistical tests
- Compute correlations for multiple variables

Now, lets continue to practice with your partner!