AN7914 Week 11 Python

May 3, 2024

1 Week 11

1.1 Data Exploration

- Descriptive statistics with pandas:
  - use of groupby for descriptives
  - custom function
- Hypothesis testing with t.test

Case-study:
- Billion Price Project: Online and Offline prices

Dataset:
- billion-prices

Import packages

[1]:
```
import numpy as np
import pandas as pd


%matplotlib inline
```

Import data

[2]:
```
bpp_original = pd.read_csv("https://osf.io/yhbr5/download", encoding="latin-1")
```

This line of code reads a CSV file named "bpp_original" from a web link. The CSV file is read using the `pd.read_csv()` function from the Pandas library in Python. The link provided is where the CSV file is located. The encoding parameter is set to "latin-1" to properly interpret the characters in the file.

Check variables

[3]:
```
bpp_original.head()
```

[3]:
```
      COUNTRY  retailer   retailer_s        date   day  month    year  \
0  ARGENTINA         1  ARGENTINA_1  2015-03-19  19.0    3.0  2015.0
1  ARGENTINA         1  ARGENTINA_1  2015-03-19  19.0    3.0  2015.0
```

```
2  ARGENTINA          1  ARGENTINA_1  2015-03-19  19.0    3.0  2015.0
3  ARGENTINA          1  ARGENTINA_1  2015-03-19  19.0    3.0  2015.0
4  ARGENTINA          1  ARGENTINA_1  2015-03-19  19.0    3.0  2015.0

              id   price  price_online  …          DEVICEID   TIME ZIPCODE  \
0   201209030113   429.0         429.0  …  891df49fb1b12aa0  21:03    8300
1  4710268235965   189.0         189.0  …  891df49fb1b12aa0  21:26    8300
2  4905524916874  6999.0        6999.0  …  891df49fb1b12aa0  21:19    8300
3  4905524925784  1999.0        2099.0  …  891df49fb1b12aa0  21:08    8300
4  4905524931310  2899.0        2899.0  …  891df49fb1b12aa0  21:06    8300

                 PHOTO OTHERSKUITEM COMMENTS PRICETYPE      CODE  sale_online  \
0  20150319_210351.jpg          NaN      NaN       NaN  124816.0          NaN
1  20150319_212653.jpg          NaN      NaN       NaN  124816.0          NaN
2  20150319_211929.jpg          NaN      NaN       NaN  124816.0          NaN
3  20150319_210847.jpg          NaN      NaN       NaN  124816.0          NaN
4  20150319_210627.jpg          NaN      NaN       NaN  124816.0          NaN

   country_s
0  Argentina
1  Argentina
2  Argentina
3  Argentina
4  Argentina

[5 rows x 21 columns]
```

Create our key variable: price differences

[4]:
```
bpp_original["p_diff"] = bpp_original["price_online"] - bpp_original["price"]
```

This line of code calculates the price difference between two columns in the "bpp_original" DataFrame: "price_online" and "price". It subtracts the "price" column from the "price_online" column and assigns the result to a new column called "p_diff".

1.2 Descriptive statistics

back

Check all the variables in DataFrame by a quick built-in summary statistics

[5]:
```
bpp_original.describe()
```

[5]:
```
           retailer           day         month          year         price  \
count  45253.000000  44928.000000  44928.000000  44928.000000  4.525300e+04
mean      34.087751     15.743523      5.301126   2015.079817  1.737368e+04
std       19.149542      8.440930      3.440339      1.035976  2.671665e+06
min        1.000000      1.000000      1.000000   2000.000000  0.000000e+00
```

```
25%       16.000000      9.000000      3.000000   2015.000000  7.000000e+00
50%       37.000000     16.000000      5.000000   2015.000000  1.999000e+01
75%       50.000000     23.000000      8.000000   2015.000000  5.799000e+01
max       62.000000     31.000000     12.000000   2016.000000  5.534910e+08

        price_online  imputed           CODE  sale_online        p_diff
count   45253.000000  22414.0   42233.000000       4144.0  4.525300e+04
mean      353.416684      1.0  181441.070253          1.0 -1.702027e+04
std      5269.492998      0.0  158106.823327          0.0  2.671661e+06
min         0.030000      1.0  112190.000000          1.0 -5.534910e+08
25%         6.990000      1.0  124816.000000          1.0  0.000000e+00
50%        19.990000      1.0  124816.000000          1.0  0.000000e+00
75%        56.990000      1.0  124816.000000          1.0  0.000000e+00
max    261690.000000      1.0  856681.000000          1.0  2.330700e+05
```

Compare key variables

[6]:
```
bpp_original.filter(["price", "price_online", "p_diff"]).describe()
```

[6]:
```
              price   price_online        p_diff
count  4.525300e+04   45253.000000  4.525300e+04
mean   1.737368e+04     353.416684 -1.702027e+04
std    2.671665e+06    5269.492998  2.671661e+06
min    0.000000e+00       0.030000 -5.534910e+08
25%    7.000000e+00       6.990000  0.000000e+00
50%    1.999000e+01      19.990000  0.000000e+00
75%    5.799000e+01      56.990000  0.000000e+00
max    5.534910e+08  261690.000000  2.330700e+05
```

Put the descriptives into columns and variables into rows

[7]:
```
bpp_original.filter(["price", "price_online", "p_diff"]).describe().transpose()
```

[7]:
```
                count          mean           std           min   25%    50%  \
price         45253.0  17373.683164  2.671665e+06  0.000000e+00  7.00  19.99
price_online  45253.0    353.416684  5.269493e+03  3.000000e-02  6.99  19.99
p_diff        45253.0 -17020.266480  2.671661e+06 -5.534910e+08  0.00   0.00

                75%          max
price         57.99  553490984.0
price_online  56.99     261690.0
p_diff         0.00     233070.0
```

Next let us check the price differences for each countries.

For this, you need to group the data and apply the required statistics to the appropriate columns

[8]:
```
bpp_original.groupby("COUNTRY").agg(
    mean_price_diff=("p_diff", "mean"), median_price_diff=("p_diff", "median")
)
```

[8]:

COUNTRY	mean_price_diff	median_price_diff
ARGENTINA	-30399.085151	0.0
AUSTRALIA	-0.464439	0.0
BRAZIL	-37.924121	0.0
CANADA	0.588671	0.0
CHINA	-0.832808	0.0
GERMANY	4.577242	0.0
JAPAN	-586.881969	0.0
SOUTHAFRICA	-125.700372	0.0
UK	-0.067043	0.0
USA	-31950.531931	0.0

Let's see what's going on with the code above.

1. `bpp_original.groupby("COUNTRY")`: This part groups the DataFrame "bpp_original" by the values in the "COUNTRY" column. After this operation, we have several smaller groups of data, each corresponding to a unique country.
2. `.agg(...)`: This is a method that aggregates the grouped data. It computes summary statistics for each group.
3. Inside the `.agg()` method:
   - `mean_price_diff=("p_diff", "mean")`: This creates a new column called "mean_price_diff" in the resulting DataFrame. It calculates the mean of the "p_diff" column within each group.
   - `median_price_diff=("p_diff", "median")`: Similarly, this creates another new column called "median_price_diff" and calculates the median of the "p_diff" column within each group.

So, this line of code computes both the mean and median of the price differences ("p_diff") for each country and stores the results in a new DataFrame. It's a concise way of summarizing data across different groups within a DataFrame.

[9]:
```
bpp_original.melt(
      id_vars=["COUNTRY"], value_vars=["price", "price_online", "p_diff"]
  )
```

[9]:

	COUNTRY	variable	value
0	ARGENTINA	price	4.290000e+02
1	ARGENTINA	price	1.890000e+02
2	ARGENTINA	price	6.999000e+03
3	ARGENTINA	price	1.999000e+03
4	ARGENTINA	price	2.899000e+03
…	…	…	…

```
135754        USA   p_diff  0.000000e+00
135755        USA   p_diff -5.534910e+08
135756        USA   p_diff  0.000000e+00
135757        USA   p_diff  0.000000e+00
135758        USA   p_diff  0.000000e+00

[135759 rows x 3 columns]
```

This line of code reshapes the DataFrame "bpp_original" using the `melt()` function from Pandas. Here's what it does:

1. `id_vars=["COUNTRY"]`: This parameter specifies which columns to keep intact without melting. In this case, it keeps the "COUNTRY" column unchanged.
2. `value_vars=["price", "price_online", "p_diff"]`: This parameter specifies which columns to melt down into a single column. It includes "price", "price_online", and "p_diff".

The `melt()` function unpivots the specified columns ("price", "price_online", and "p_diff") into a long format, with each row representing a unique combination of "COUNTRY" and one of the specified columns. The "variable" column contains the original column names, and the "value" column contains the corresponding values.

Lets say we are interested in the prices as well for each countries.

[10]:
```
(
    bpp_original.melt(
        id_vars=["COUNTRY"], value_vars=["price", "price_online", "p_diff"]
    )
    .groupby(["COUNTRY", "variable"])
    .agg(Mean=("value", "mean"), Median=("value", "median"))
)
```

[10]:

		Mean	Median
COUNTRY	variable		
ARGENTINA	p_diff	-30399.085151	0.00
	price	31061.999723	54.95
	price_online	662.914572	55.00
AUSTRALIA	p_diff	-0.464439	0.00
	price	22.126683	7.99
	price_online	21.662243	8.00
BRAZIL	p_diff	-37.924121	0.00
	price	338.507332	69.90
	price_online	300.583211	67.90
CANADA	p_diff	0.588671	0.00
	price	35.799147	17.98
	price_online	36.387818	17.99
CHINA	p_diff	-0.832808	0.00
	price	141.923942	43.85
	price_online	141.091135	43.90

```
GERMANY      p_diff            4.577242     0.00
             price            31.831955    14.99
             price_online     36.409198    15.99
JAPAN        p_diff         -586.881969     0.00
             price          5508.326655  1180.00
             price_online   4921.444686   973.00
SOUTHAFRICA  p_diff         -125.700372     0.00
             price           208.401621    39.99
             price_online     82.701249    39.99
UK           p_diff           -0.067043     0.00
             price             7.841262     2.00
             price_online      7.774219     2.00
USA          p_diff       -31950.531931     0.00
             price         31985.195144    14.99
             price_online     34.663213    14.99
```

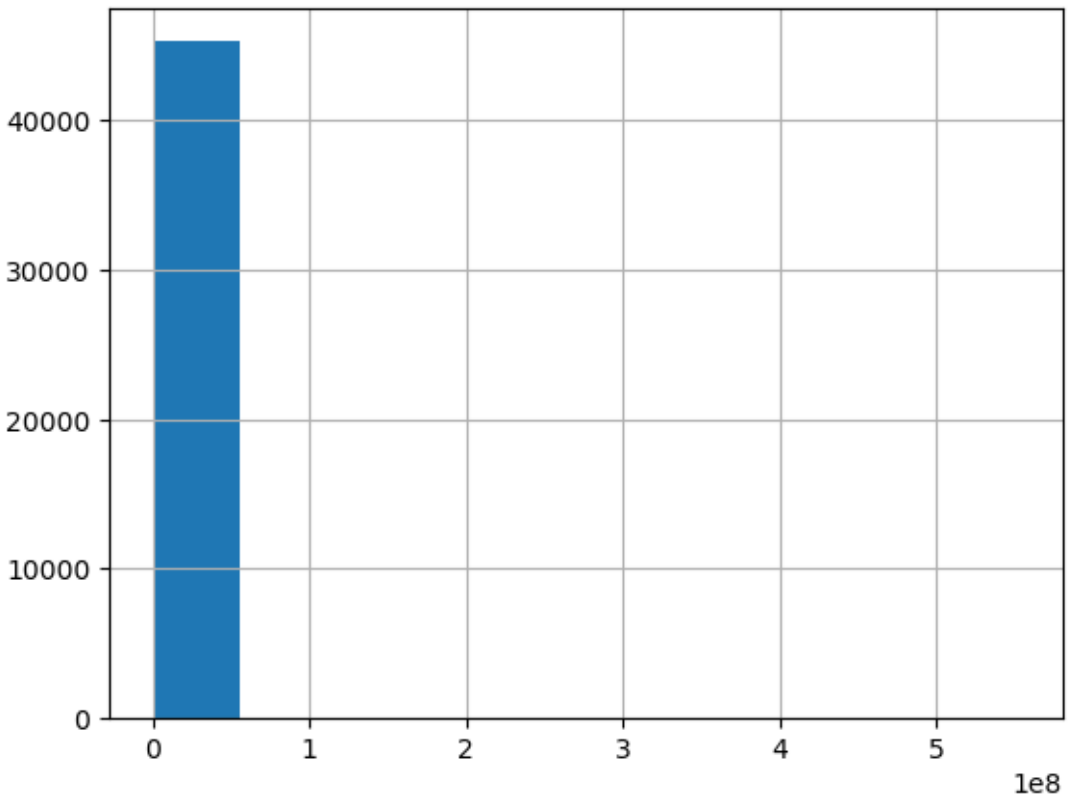
This code first melts the DataFrame "bpp_original" to reshape it, then groups the melted DataFrame by both "COUNTRY" and "variable". After that, it calculates two aggregate statistics for the "value" column within each group: the mean and median. The results are stored in a new DataFrame with columns named "Mean" and "Median".

Check the empirical distribution: histogram.

look at the histogram using thew built in pandas `hist()` function

[11]:
```
bpp_original.price.hist()
```

[11]:
```
<AxesSubplot: >
```



It is clear: need to filter out some data!

We can do this in two ways and they both gives us the same result:

[12]:
```
bpp = (
    bpp_original.loc[bpp_original["sale_online"].isnull()]
    .loc[bpp_original["price"].notnull()]
    .loc[bpp_original["price_online"].notnull()]
    .loc[bpp_original["PRICETYPE"] == "Regular Price"]
)
```

or you can do this

[13]:
```
bpp1 = bpp_original[
    (bpp_original["sale_online"].isnull()) &
    (bpp_original["price"].notnull()) &
    (bpp_original["price_online"].notnull()) &
    (bpp_original["PRICETYPE"] == "Regular Price")
]
```

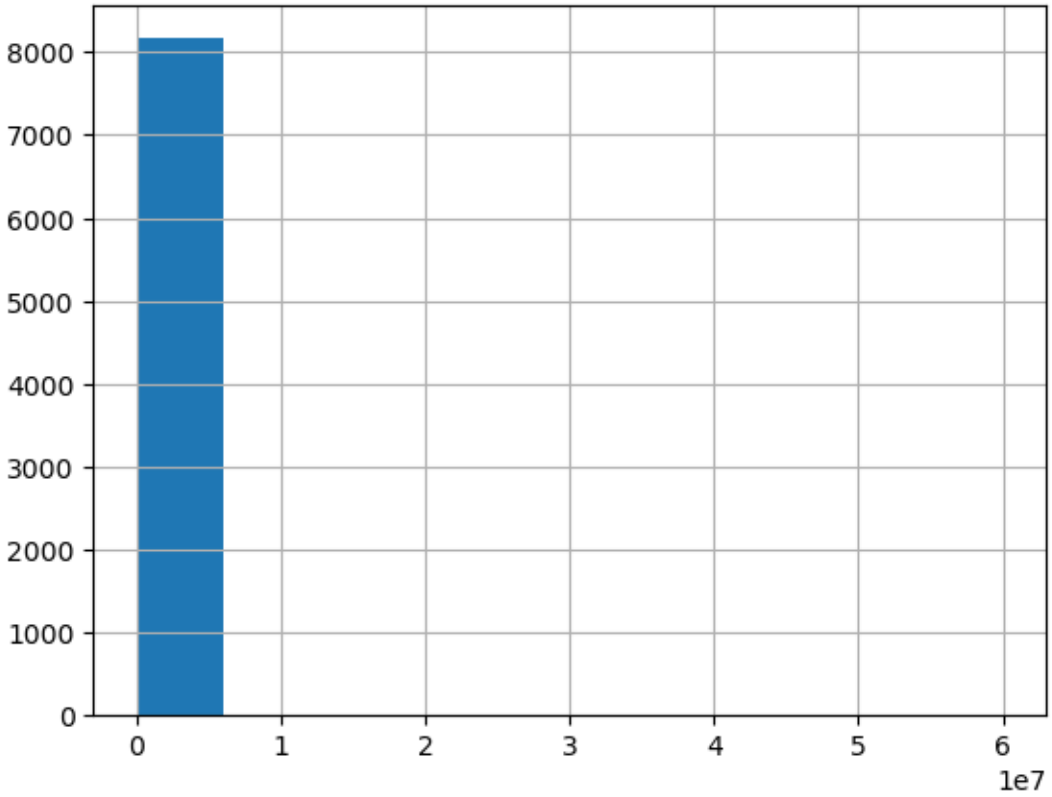
This line of code creates a new DataFrame called “bpp1” by filtering rows from the “bpp_original” DataFrame based on multiple conditions:

1. `bpp_original["sale_online"].isnull()`: This condition checks if the "sale_online" column is null.
2. `bpp_original["price"].notnull()`: This condition checks if the "price" column is not null.
3. `bpp_original["price_online"].notnull()`: This condition checks if the "price_online" column is not null.
4. `bpp_original["PRICETYPE"] == "Regular Price"`: This condition checks if the value in the "PRICETYPE" column is "Regular Price".

The DataFrame "bpp1" will contain only the rows from "bpp_original" that satisfy all of these conditions.

[14]:
```
bpp.price.hist()
```

[14]:
```
<AxesSubplot: >
```



Check our newly created data:

[15]:
```
bpp.filter(["price", "price_online", "p_diff"]).describe()
```

[15]:

	price	price_online	p_diff
count	8.169000e+03	8169.00000	8.169000e+03

```
mean   7.650828e+03     133.36461 -7.517463e+03
std    6.641562e+05     495.47564  6.641574e+05
min    2.500000e-01       0.25000 -6.002112e+07
25%    5.990000e+00       5.99000 -1.000000e-01
50%    1.499000e+01      14.99000  0.000000e+00
75%    4.399000e+01      44.95000  0.000000e+00
max    6.002113e+07    6362.00000  9.200100e+02
```

Drop obvious errors: price is larger than $1000

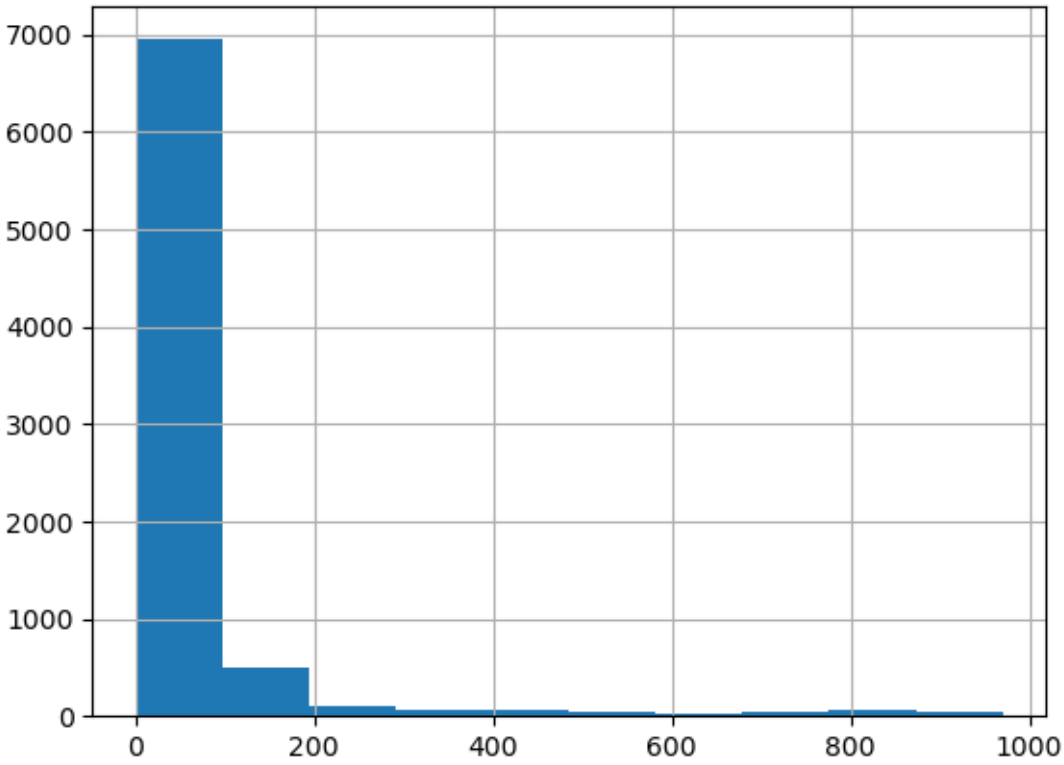
[16]:
```
bpp = bpp.loc[bpp["price"] < 1000]
```

[17]:
```
bpp.filter(["price", "price_online", "p_diff"]).describe()
```

[17]:
```
              price  price_online       p_diff
count  7893.000000   7893.000000  7893.000000
mean     55.211356     54.913554    -0.297802
std     135.469561    134.315549    20.141510
min       0.250000      0.250000  -380.130000
25%       5.990000      5.790000     0.000000
50%      14.490000     13.990000     0.000000
75%      38.190000     39.000000     0.000000
max     970.000000    970.000000   920.010000
```

[18]:
```
bpp.price.hist()
```

[18]:
```
<AxesSubplot: >
```



1.2.1 Hypothesis testing

back

Test 1:

H0: the average price difference between price_online - price = 0
HA: the avg price diff is non 0.

[19]:
```
from scipy import stats
```

[20]:
```
stats.ttest_1samp(bpp["p_diff"], 0)
```

[20]:
```
TtestResult(statistic=-1.3135807936650183, pvalue=0.18902550071796193, df=7892)
```

Test 2: The online prices are greater or equal to offline prices

H0: price_online - price = 0
HA: price_online - price > 0

[21]:
```
stats.ttest_1samp(bpp["p_diff"], 0, alternative="greater")
```

[21]:
```
TtestResult(statistic=-1.3135807936650183, pvalue=0.905487249641019, df=7892)
```

Test 3: The online prices are smaller or equal to offline prices

H0: price_online - price = 0
HA: price_online - price < 0

[22]:
```
stats.ttest_1samp(bpp["p_diff"], 0, alternative="less")
```

[22]:
```
TtestResult(statistic=-1.3135807936650183, pvalue=0.09451275035898096, df=7892)
```

Let us create multiple hypothesis tests:
Check the hypothesis that online prices are the same as offline for each country!

[23]:
```
testing = bpp.groupby("COUNTRY").agg(
    mean_pdiff=("p_diff", "mean"),
    se_pdiff=("p_diff", "sem"),
    num_obs=("p_diff", "count"),
)
testing
```

[23]:

COUNTRY	mean_pdiff	se_pdiff	num_obs
BRAZIL	-0.905328	0.784719	122
CHINA	-0.510526	0.841118	19
GERMANY	7.065190	3.102340	422
JAPAN	-11.982857	2.146688	350
SOUTHAFRICA	-2.529723	0.831934	541
USA	0.054460	0.124552	6439

Testing is easy if one understands the theory!
t_stat: with this H0 and t-test:

[24]:
```
testing["t_stat"] = testing["mean_pdiff"] / testing["se_pdiff"]

testing
```

[24]:

COUNTRY	mean_pdiff	se_pdiff	num_obs	t_stat
BRAZIL	-0.905328	0.784719	122	-1.153697
CHINA	-0.510526	0.841118	19	-0.606962
GERMANY	7.065190	3.102340	422	2.277374
JAPAN	-11.982857	2.146688	350	-5.582022
SOUTHAFRICA	-2.529723	0.831934	541	-3.040772
USA	0.054460	0.124552	6439	0.437248

Calculate p-values

[25]:
```
testing["p_val"] = stats.t.sf(abs(testing["t_stat"]), df=testing["num_obs"] - 1)

testing
```

[25]:

COUNTRY	mean_pdiff	se_pdiff	num_obs	t_stat	p_val
BRAZIL	-0.905328	0.784719	122	-1.153697	1.254490e-01
CHINA	-0.510526	0.841118	19	-0.606962	2.757282e-01
GERMANY	7.065190	3.102340	422	2.277374	1.163233e-02
JAPAN	-11.982857	2.146688	350	-5.582022	2.390995e-08
SOUTHAFRICA	-2.529723	0.831934	541	-3.040772	1.237115e-03
USA	0.054460	0.124552	6439	0.437248	3.309730e-01

Round it to 4 digits

[26]:
```
testing["p_val"] = testing["p_val"].round(4)
testing
```

[26]:

COUNTRY	mean_pdiff	se_pdiff	num_obs	t_stat	p_val
BRAZIL	-0.905328	0.784719	122	-1.153697	0.1254
CHINA	-0.510526	0.841118	19	-0.606962	0.2757
GERMANY	7.065190	3.102340	422	2.277374	0.0116
JAPAN	-11.982857	2.146688	350	-5.582022	0.0000
SOUTHAFRICA	-2.529723	0.831934	541	-3.040772	0.0012
USA	0.054460	0.124552	6439	0.437248	0.3310