Bin the maximum temperature data, separately for the two stations, using the categories Ranson used

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from datetime import datetime
from scipy.stats import chisquare
from func5 import get_stat, find_p_value, fisher_comb
```

In [2]:
```
#load station_adjusted.csv from group assignment 3, which contains bias cor
#for all stations from 1980 to 2009 using categories Ranson used
station_adjusted = pd.read_csv("../group_assignment3/station_adjusted.csv")
```

```
/Users/glance/anaconda3/lib/python3.6/site-packages/IPython/core/interact
iveshell.py:3020: DtypeWarning: Columns (10,19) have mixed types. Specify
dtype option on import or set low_memory=False.
  interactivity=interactivity, compiler=compiler, result=result)
```

In [3]:
```
# filter out HCN Berkeley station (ID: USC00040693) and the HCN Livermore st
Berkeley = station_adjusted.loc[station_adjusted['ID'] == 'USC00040693']
Berkeley.head()
```

Out[3]:

	Unnamed: 0	Unnamed: 0_x	Unnamed: 0.1	ID	LATITUDE	LONGITUDE	ELEVATION	STATE
2953	2953	1117	80982	USC00040693	37.8744	-122.2606	94.5	CA
2954	2954	1117	80982	USC00040693	37.8744	-122.2606	94.5	CA
2955	2955	1117	80982	USC00040693	37.8744	-122.2606	94.5	CA
2956	2956	1117	80982	USC00040693	37.8744	-122.2606	94.5	CA
2957	2957	1117	80982	USC00040693	37.8744	-122.2606	94.5	CA

5 rows × 22 columns

In [4]:
```
Livermore = station_adjusted.loc[station_adjusted['ID'] == 'USC00044997']
Livermore.head()
```

Out[4]:

	Unnamed: 0	Unnamed: 0_x	Unnamed: 0.1	ID	LATITUDE	LONGITUDE	ELEVATION	STATE
33156	33156	1626	81491	USC00044997	37.6922	-121.7692	146.3	CA
33157	33157	1626	81491	USC00044997	37.6922	-121.7692	146.3	CA
33158	33158	1626	81491	USC00044997	37.6922	-121.7692	146.3	CA
33159	33159	1626	81491	USC00044997	37.6922	-121.7692	146.3	CA
33160	33160	1626	81491	USC00044997	37.6922	-121.7692	146.3	CA

5 rows × 22 columns

In [5]:
```
# use inverse distance to calculate aggregate max temperature for Berkeley s
Berkeley_TMAX = Berkeley[Berkeley['ELEMENT'] == 'TMAX']
Berkeley_TMAX['wi_val'] = Berkeley_TMAX['INVDIST'] * Berkeley_TMAX['DATA VAI
weighted_Berkeley_TMAX = Berkeley_TMAX.groupby('YEARMONTHDAY', as_index = Fa
dict_Berkeley_TMAX = {}
for i in Berkeley_TMAX['YEARMONTHDAY']:
    only_i = Berkeley_TMAX[Berkeley_TMAX['YEARMONTHDAY'] == i]
    dict_Berkeley_TMAX[i] = sum(only_i['wi_val'])/sum(only_i['INVDIST'].uniq

#check if data makes sense
series_Berkeley_TMAX = pd.Series(data = dict_Berkeley_TMAX)
series_Berkeley_TMAX.describe()
```

```
/Users/glance/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.p
y:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-doc
s/stable/indexing.html#indexing-view-versus-copy (http://pandas.pydata.or
g/pandas-docs/stable/indexing.html#indexing-view-versus-copy)
  This is separate from the ipykernel package so we can avoid doing impor
ts until
```

Out[5]:
```
count    9409.000000
mean       68.962727
std         8.837935
min        40.587580
25%        62.727580
50%        68.667580
75%        74.607580
max       109.707580
dtype: float64
```

In [6]:
```
# use inverse distance to calculate aggregate max temperature for Livermore
Livermore_TMAX = Livermore[Livermore['ELEMENT'] == 'TMAX']
Livermore_TMAX['wi_val'] = Livermore_TMAX['INVDIST'] * Livermore_TMAX['DATA
weighted_Livermore_TMAX = Livermore_TMAX.groupby('YEARMONTHDAY', as_index =
dict_Livermore_TMAX = {}
for i in Livermore_TMAX['YEARMONTHDAY']:
    only_i = Livermore_TMAX[Livermore_TMAX['YEARMONTHDAY'] == i]
    dict_Livermore_TMAX[i] = sum(only_i['wi_val'])/sum(only_i['INVDIST'].uni

#check if data makes sense
series_Livermore_TMAX = pd.Series(data = dict_Livermore_TMAX)
series_Livermore_TMAX.describe()
```

```
/Users/glance/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.p
y:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-doc
s/stable/indexing.html#indexing-view-versus-copy (http://pandas.pydata.or
g/pandas-docs/stable/indexing.html#indexing-view-versus-copy)
  This is separate from the ipykernel package so we can avoid doing impor
ts until
```

Out[6]:
```
count    10863.000000
mean        68.807978
std         14.033232
min         21.573694
25%         57.573694
50%         67.473694
75%         79.533694
max        107.433694
dtype: float64
```

Bin for Berkeley:

In [7]:
```
# Bin TMAX for Berkeley
import math
#define bins
TMAX_Berkeley_df = pd.DataFrame({'DATE':series_Berkeley_TMAX.index, 'TMAX':s
temp_bins = [-math.inf,10,20,30,40,50,60,70,80,90,100,math.inf]
group_names_temp = ['<10F','10-19F','20-29F','30-39F','40-49F','50-59F','60-
TMAX_Berkeley_df['temp_bins'] = pd.cut(TMAX_Berkeley_df['TMAX'], temp_bins,
TMAX_Berkeley_df.head()
```

Out[7]:

	DATE	TMAX	temp_bins
0	19800101.0	61.64758	60-69F
1	19800102.0	56.60758	50-59F
2	19800103.0	58.58758	50-59F
3	19800104.0	57.68758	50-59F
4	19800105.0	58.58758	50-59F

In [8]:
```
TMAX_Berkeley_df['YearMonth'] = TMAX_Berkeley_df['DATE'].astype(str).str[:6]
TMAX_Berkeley_df.head(10)
```

Out[8]:

	DATE	TMAX	temp_bins	YearMonth
0	19800101.0	61.64758	60-69F	198001
1	19800102.0	56.60758	50-59F	198001
2	19800103.0	58.58758	50-59F	198001
3	19800104.0	57.68758	50-59F	198001
4	19800105.0	58.58758	50-59F	198001
5	19800106.0	57.68758	50-59F	198001
6	19800107.0	62.72758	60-69F	198001
7	19800108.0	59.66758	50-59F	198001
8	19800109.0	56.60758	50-59F	198001
9	19800110.0	51.56758	50-59F	198001

In [9]:
```
#count frequancy of bin for Berkeley TMAX in each month
TMAX_Berkeley_pivot = TMAX_Berkeley_df.pivot_table(index='temp_bins',
                        columns='YearMonth',
                        values='TMAX',
                        fill_value=0,
                        aggfunc='count').unstack().to_frame().sort_index()
TMAX_Berkeley_pivot.head(20)
```

Out[9]:

		0
YearMonth	**temp_bins**	
198001	**40-49F**	0
	50-59F	19
	60-69F	12
	70-79F	0
	80-89F	0
	90-99F	0
	>100F	0
198002	**40-49F**	0
	50-59F	3
	60-69F	26
	70-79F	0
	80-89F	0
	90-99F	0
	>100F	0
198003	**40-49F**	0
	50-59F	4
	60-69F	21
	70-79F	6
	80-89F	0
	90-99F	0

In [10]:
```
#save to csv.file
TMAX_Berkeley_pivot.to_csv('TMAX_Berkeley_pivot.csv')
```

Bin for Livermore:

In [11]:
```
# Similarly, bin TMAX for Livermore
TMAX_Livermore_df = pd.DataFrame({'DATE':series_Livermore_TMAX.index, 'TMAX'
temp_bins = [-math.inf,10,20,30,40,50,60,70,80,90,100,math.inf]
group_names_temp = ['<10F','10-19F','20-29F','30-39F','40-49F','50-59F','60-
TMAX_Livermore_df['temp_bins'] = pd.cut(TMAX_Livermore_df['TMAX'], temp_bins
TMAX_Livermore_df.tail()
```

Out[11]:

	DATE	TMAX	temp_bins
10858	20091227.0	50.553694	50-59F
10859	20091228.0	44.433694	40-49F
10860	20091229.0	49.473694	40-49F
10861	20091230.0	54.513694	50-59F
10862	20091231.0	55.593694	50-59F

In [12]:
```
TMAX_Livermore_df['YearMonth'] = TMAX_Livermore_df['DATE'].astype(str).str[
```

In [13]:
```
TMAX_Livermore_df.head()
```

Out[13]:

	DATE	TMAX	temp_bins	YearMonth
0	19800101.0	56.493694	50-59F	198001
1	19800102.0	54.513694	50-59F	198001
2	19800103.0	50.553694	50-59F	198001
3	19800104.0	44.433694	40-49F	198001
4	19800105.0	49.473694	40-49F	198001

In [14]:
```
#count frequancy of bin for Berkeley TMAX in each month
TMAX_Livermore_pivot = TMAX_Livermore_df.pivot_table(index='temp_bins',
                        columns='YearMonth',
                        values='TMAX',
                        fill_value=0,
                        aggfunc='count').unstack().to_frame().sort_index()
```

In [15]:
```
TMAX_Livermore_pivot.head(20)
```

Out[15]:

YearMonth	temp_bins	0
198001	20-29F	0
	30-39F	0
	40-49F	9
	50-59F	20
	60-69F	2
	70-79F	0
	80-89F	0
	90-99F	0
	>100F	0
198002	20-29F	0
	30-39F	0
	40-49F	0
	50-59F	19
	60-69F	10
	70-79F	0
	80-89F	0
	90-99F	0
	>100F	0
198003	20-29F	0
	30-39F	0

In [16]:
```
#save to csv.file
TMAX_Livermore_pivot.to_csv('TMAX_Livermore_pivot.csv')
```

Devise and implement a stratified permutation test for the hypothesis that the two cities have "the same weather."

Formulate the hypothesis as a generalized two-sample problem, i.e., ask whether differences (between the cities) in the number of days each month in which the maximum temperature is in each bin could reasonably be attributed to chance, if the maximum temperatures had been a single population of numbers randomly split across the two cities.

What did you stratify on? Why is that a good choice? Why stratify at all?

Answer: We stratify on month. First, it is fundamental to stratify on month & year, because different seasons (months) are known to have different temperatures and particular years may be hotter/colder than others. Also, monthly numbers are what Ranson uses as input. So we think it is a good choice to randomize by day and stratify on month to make the comparison fair enough.

Answer: We need to stratify because weather follows seasonal patterns. Summer tends to have higher temperature and winner tends to have lower ones. Therefore, random allocation of entire data into two groups is unreasonable. It is unlikely for temperature of everyday of every year to be equally likely to end up on Berkeley or Livermore. Randomizing by day (flipped a coin to give a particular day’s weather to Berkeley or Livermore) and stratifing on month help us make more reasonable comparisons.

Permutation test:

Our permutation tests are performed on each month from 1980 to 2009, which is the time period of weather data. We constructed Intersection-Union Hypotheses as following:

Null hypothesis: "The maximum temperatures had been a single population of numbers randomly split across Berkeley and Livermore"

It can be written as an intersection of hypotheses:
(The maximum temperatures had been a single population of numbers randomly split across Berkeley and Livermore in Jan, 1980) $\cap$ (... in Feb, 1980) $\cap$ (... in Mar, 1980) $\cap \cdots \cap$ (... in Dec, 2009).

Alternative hypothesis: "The maximum temperatures of Berkeley and Livermore had been two different populations of numbers"

It can be written as (The maximum temperatures of Berkeley and Livermore had been two different populations of numbers in Jan, 1980) $\cap$ (... in Feb, 1980) $\cap$ (... in Mar, 1980) $\cap \cdots \cap$ (... in Dec, 2009).

In [17]:
```
#check how many unique YearMonths there are for Berkeley:
len(TMAX_Berkeley_df['DATE'].unique())
```

Out[17]: 9409

In [18]:
```
#check how many unique YearMonths there are for Livermore:
len(TMAX_Livermore_df['DATE'].unique())
```

Out[18]: 10863

Since Berkeley and Livermore have available data for different YearMonths, we only look at YearMonths when both of them have weather data:

In [19]:
```
#YearMonths when both of them have weather data:
shared_dates = list(set(TMAX_Berkeley_df['DATE'].unique()) & set(TMAX_Livern
```

In [20]:
```
len(shared_dates)
```

Out[20]: 9314

In [21]:

```
#Extract weather data only from shared_dates for Berkeley:
mask_ber = [i in shared_dates for i in TMAX_Berkeley_df['DATE']]
Berkeley_df = TMAX_Berkeley_df[mask_ber]
Berkeley_df.head()
```

Out[21]:

	DATE	TMAX	temp_bins	YearMonth
0	19800101.0	61.64758	60-69F	198001
1	19800102.0	56.60758	50-59F	198001
2	19800103.0	58.58758	50-59F	198001
3	19800104.0	57.68758	50-59F	198001
4	19800105.0	58.58758	50-59F	198001

In [22]:

```
#check the length
len(Berkeley_df['DATE'].unique())
```

Out[22]: 9314

In [23]:

```
#same thing for Livermore:
mask_liv = [i in shared_dates for i in TMAX_Livermore_df['DATE']]
Livermore_df = TMAX_Livermore_df[mask_liv]
Livermore_df.head()
```

Out[23]:

	DATE	TMAX	temp_bins	YearMonth
0	19800101.0	56.493694	50-59F	198001
1	19800102.0	54.513694	50-59F	198001
2	19800103.0	50.553694	50-59F	198001
3	19800104.0	44.433694	40-49F	198001
4	19800105.0	49.473694	40-49F	198001

In [24]:

```
#check the length
len(Livermore_df['DATE'].unique())
```

Out[24]: 9314

In [25]:
```
#pivot table of Berkeley's grouped counts of days by 11 temp_bins for each Y
Berkeley_grouped_counts = Berkeley_df.pivot_table(index=['temp_bins', 'YearM
                    values='TMAX',
                    fill_value=0,
                    aggfunc='count',
                    dropna= False).unstack().sort_index()
Berkeley_grouped_counts
```

Out[25]:

	TMAX											
YearMonth	198001	198002	198003	198004	198005	198006	198007	198008	198009	198010	...	2
temp_bins												
<10F	0	0	0	0	0	0	0	0	0	0	...	
10-19F	0	0	0	0	0	0	0	0	0	0	...	
20-29F	0	0	0	0	0	0	0	0	0	0	...	
30-39F	0	0	0	0	0	0	0	0	0	0	...	
40-49F	0	0	0	0	0	0	0	0	0	0	...	
50-59F	19	3	4	4	3	0	0	1	0	0	...	
60-69F	12	26	21	20	23	18	13	20	15	12	...	
70-79F	0	0	6	5	5	10	13	9	10	14	...	
80-89F	0	0	0	1	0	2	5	1	4	1	...	

In [26]:
```
#pivot table of Livermore's grouped counts of days by 11 temp_bins for each
Livermore_grouped_counts = Livermore_df.pivot_table(index=['temp_bins', 'Yea
                    values='TMAX',
                    fill_value=0,
                    aggfunc='count',
                    dropna= False).unstack().sort_index()
Livermore_grouped_counts
```

Out[26]:

	TMAX											
YearMonth	198001	198002	198003	198004	198005	198006	198007	198008	198009	198010	...	2
temp_bins												
<10F	0	0	0	0	0	0	0	0	0	0	...	
10-19F	0	0	0	0	0	0	0	0	0	0	...	
20-29F	0	0	0	0	0	0	0	0	0	0	...	
30-39F	0	0	0	0	0	0	0	0	0	0	...	
40-49F	9	0	0	1	0	0	0	0	0	0	...	
50-59F	20	19	16	5	4	0	0	0	0	0	...	
60-69F	2	10	14	13	14	10	4	0	2	10	...	
70-79F	0	0	1	9	9	11	6	8	11	10	...	
80-89F	0	0	0	2	3	8	8	17	13	2	...	

To calculate our observed statistic: sum（Berkbins i-livbins i）^2, we did the following matrix(dateframe) computing:

In [27]:
```
# (Berkbins i-livbins i) ^2
diff_squared = Berkeley_grouped_counts.subtract(Livermore_grouped_counts)**2
diff_squared
```

Out[27]:

	TMAX										
YearMonth	198001	198002	198003	198004	198005	198006	198007	198008	198009	198010	...
temp_bins											
<10F	0	0	0	0	0	0	0	0	0	0	...
10-19F	0	0	0	0	0	0	0	0	0	0	...
20-29F	0	0	0	0	0	0	0	0	0	0	...
30-39F	0	0	0	0	0	0	0	0	0	0	...
40-49F	81	0	0	1	0	0	0	0	0	0	...
50-59F	1	256	144	1	1	0	0	1	0	0	...
60-69F	100	256	49	49	81	64	81	400	169	4	...
70-79F	0	0	25	16	16	1	49	1	1	16	...
80-89F	0	0	0	1	9	36	9	256	81	1	...
90-99F	0	0	0	0	1	1	81	25	9	9	...
>100F	0	0	0	0	0	0	16	1	0	4	...

11 rows × 312 columns

In [28]:
```
#for each unique YearMonth, sum the squared difference, and put into datafra
obs_stats_by_months = diff_squared.sum(0)
pd.DataFrame(obs_stats_by_months).head()
```

Out[28]:

		0
	YearMonth	
TMAX	198001	182
	198002	512
	198003	218
	198004	68
	198005	108

In [29]:
```
#create a table for observed and permutated statistics:
stats_table = pd.DataFrame(obs_stats_by_months)
```

In [30]:
```
#change column name
stats_table.columns = ['observed']
```

We wrote the above process of getting statistics into a function called 'get_stat()' for later use, please see func5.py for details.

Permutation (1000 times):

In [31]:
```
#set a random seed
np.random.seed(101)


our_stats = stats_table

#Get the temp_bins for Berkeley and Livermore:
Bins_Ber = Berkeley_df['temp_bins']
Bins_Liv = Livermore_df['temp_bins']
two_bins = [(i, j) for i, j in zip(Bins_Ber, Bins_Liv)]

for i in range(1000):
    #For each day, shuffle the bins for Berkeley and Livermore randomly:
    permutated_bins = [np.random.permutation(i) for i in two_bins]

    #create a column called 'per_bins' for the shuffled bins:
    Berkeley_df['per_bins'] = [i[0] for i in permutated_bins]
    Livermore_df['per_bins'] = [i[1] for i in permutated_bins]

    #for two cities, get pivot table of grouped counts of days by 11 temp_bi
    Berkeley_per_counts = Berkeley_df.pivot_table(index=['per_bins', 'YearMo
                                                  values='TMAX',
                                                  fill_value=0,
                                                  aggfunc='count',
                                                  dropna= False).unstack()

    Livermore_per_counts = Livermore_df.pivot_table(index=['per_bins', 'Year
                                                  values='TMAX',
                                                  fill_value=0,
                                                  aggfunc='count',
                                                  dropna= False).unstack()

    #Calculate a column of test statistics for each permutation and append i
    #please see func5.py for more detail of 'get_stat()'
    per_stat = get_stat(Berkeley_per_counts, Livermore_per_counts)
    per_stat.columns = ['permutation No.' + str(i)]
    our_stats = pd.concat([our_stats, per_stat], axis=1)
```

```
/Users/glance/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.p
y:17: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-doc
s/stable/indexing.html#indexing-view-versus-copy (http://pandas.pydata.or
g/pandas-docs/stable/indexing.html#indexing-view-versus-copy)
/Users/glance/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.p
y:18: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-doc
s/stable/indexing.html#indexing-view-versus-copy (http://pandas.pydata.or
g/pandas-docs/stable/indexing.html#indexing-view-versus-copy)
```

In [32]:
```
#our statistics table:
our_stats.head()
```

Out[32]:

	YearMonth	observed	permutation No.0	permutation No.1	permutation No.2	permutation No.3	permutation No.4	peri No.
TMAX	**198001**	182	14	18	14.0	54	14	
	198002	512	8	32	72.0	128	8	
	198003	218	6	74	6.0	42	62	
	198004	68	16	8	52.0	8	28	
	198005	108	12	16	28.0	4	8	

5 rows × 1001 columns

In [33]:
```
#get a list of p-values, each p-value is for one unique month, given our sta
#please see func5.py for more detail of 'find_p_value()'
p_values = find_p_value(our_stats)
```

In [34]:
```
p_values[:10]
```

Out[34]:
```
[0.017982017982017984,
 0.000999000999000999,
 0.005994005994005994,
 0.023976023976023976,
 0.022977022977022976,
 0.03196803196803197,
 0.000999000999000999,
 0.000999000999000999,
 0.000999000999000999,
 0.37662337662337664]
```

In [35]:
```
#check the length
len(p_values)
```

Out[35]: 312

Combine results across strata using Fisher's combining function:

$$\phi_F(\lambda) \equiv -2 \sum_{j=1}^{n} \ln(\lambda_j).$$

In [36]:
```
#please see func5.py for more detail of 'fisher_comb()'
fisher_comb(p_values)
```

Out[36]: 2532.167669482353

Conclusion:

Chisquare with df = 2*312, p = 0.05: a number between 658.094 to 710.421
What we got from fisher combining -- 2532.167669482353 is surely larger than such a number, so we reject the null hypothesis that maximum temperatures had been a single population of numbers randomly split across Berkeley and Livermore.

Can you use the chi-square distribution to calibrate the test? Why or why not?

Answer: No. Because we rely on randomization in calibrating how surprising we should be to see the observed difference, instead of relying on something involving chi-square distribution.

Discuss how to take into account simulation uncertainty in estimating the overall P-value:

Here, we use 'np.random', which is also considered to be a PRNG mentioned in the lecture,
According to https://github.com/pbstark/S157F17/blob/master/combining-tests.ipynb (https://github.com/pbstark/S157F17/blob/master/combining-tests.ipynb),
"We will pretend that the simulation itself is perfect: that the PRNG generates true IID $U[0, 1]$ variables, that pseudo-random integers on $\{0, 1, \ldots, N\}$ really are equally likely, and that pseudo-random samples or permutations really are equally likely, etc. The error we are accounting for is not the imperfection of the PRNG or other algorithms, just the uncertainty due to approximating a theoretical probability λ_j by an estimate via (perfect) simulation."

Discuss what this means for Ranson's approach

Ranson's semi-parametric method of binning intends to prevent saying that every degree increase in temperature has the same effect on crime rate. However, our permutation test indicates that the weather from two cities in the same county significantly differ from each other, implying that every degree increase in different cities might have different impact on crime rate. Thus, it is not appropriate to average temperature by county, Ranson's method of binning might also lose its meaning when using averaged temperature by county.