assignment_1(NS)

August 18, 2018

1 Mandatory Assignment 1

This is the first of two mandatory assignments which must be completed during the course. First some practical information:

- When is the assignment due?: **23:59, Sunday, August 19, 2018.**
- How do you grade the assignment?: You will **peergrade** each other as primary grading.
- Can i work with my group?: **yes**

The assigment consist of one to tree problems from each of the exercise sets you have solved so far (excluding set 1 and set 9). We've tried to select problems which are self contained, but it might be nessecary to solve some of the previous exercises in each set to fully answer the problems in this assignment.

1.1 Problems from Exercise Set 2:

Ex. 2.2: Make two lists. The first should be numbered. The second should be unnumbered and contain at least one sublevel.

[Answer to Ex. 2.2 here] 1. Hello 2. There 3. Friend

- This
- Is
  - A
    * List

1.2 Problems from Exercise set 3:

Ex. 3.1.3: Let `l1 = ['r ', 'Is', '>', ' < ', 'g ', '?']`. Create from `l1` the sentence "Is r > g?" using your knowledge about string formatting. Make sure there is only one space in between worlds.

> *Hint:* You should be able to combine the above informations to solve this exercise.

```
In [31]: # [Answer to Ex. 3.1.3 here]
         l1 = ['r ', 'Is', '>', ' < ', 'g ', '?']
         #" ".join([l1[1],l1[0],l1[2],l1[-2]]) + l1[-1]
             #Using the .join command would've been neat,
             #but due to the random white spaces in the strings it won't work that simply.
         l1[1] + " " + l1[0] + l1[2] +" " + l1[-2].strip() + l1[-1]

Out[31]: 'Is r > g?'
```

Ex. 3.1.4: Create an empty dictionary `words` using the `dict()`function. Then add each of the words in `['animal', 'coffee', 'python', 'unit', 'knowledge', 'tread', 'arise']` as a key, with the value being a boolean indicator for whether the word begins with a vowel. The results should look like `{'bacon': False, 'asynchronous': True ...}`

Hint: You might want co first construct a function that asseses whether a given word begins with a vowel or not.

```
In [32]: # [Answer to Ex. 3.1.4 here]
         words = dict()
         dict_list1 = ['animal', 'coffee', 'python', 'unit', 'knowledge', 'tread', 'arise']
         dict_list2 = []
         for key in dict_list1: #A function for identifying vowels and appending them is creat
             if key[0] in 'aeiou': #The first letter of each string in the list is checked
                 dict_list2.append(True) #If the letter is a vowel, append a True to the list
             else:
                 dict_list2.append(False) #If not, append a False to the list.
         print(dict_list2) #the results of the looped are examined and it looks good!
             #dict_list2 = [True,False,False,True,False,False,True] #This is how the list shou

         dict_zip =  list(zip(dict_list1, dict_list2)) #The two lists are zipped together.
         words.update(dict_zip) #And then added to the dictionary 'words'
         words

[True, False, False, True, False, False, True]


Out[32]: {'animal': True,
          'coffee': False,
          'python': False,
          'unit': True,
          'knowledge': False,
          'tread': False,
          'arise': True}
```

Ex. 3.3.2: use the `requests` module (get it with `pip install requests`) and `construct_link()` to request birth data from the "FOD" table. Get all available years (variable "Tid"), but only female births (BARNKON=P) . Unpack the json payload and store the result. Wrap the whole thing in a function which takes an url as input and returns the corresponding output.

Hint: The `requests.response` object has a `.json()` method.

Note: you wrote `construct_link()` in 3.3.1, if you didn't heres the link you need to get: `https://api.statbank.dk/v1/data/FOLK1A/JSONSTAT?lang=en&Tid=*`

In [33]:
```
# [Answer to Ex. 3.3.2 here]
import requests
import re
#A function which retrieves a desired URL as json
def get_data(url):
    data_json=requests.get(url).json() #Fetch the desired url as a json data.
    return data_json


get_data('https://api.statbank.dk/v1/data/FOD/JSONSTAT?lang=en&Tid=*&BARNKON=P')

#a function which retrieves the data from a table-input and the desired variabel-spec
def get_data(table_id,variables):
    base = 'https://api.statbank.dk/v1/data/{id}/JSONSTAT?lang=en'.format(id = table_

    for var in variables:
        base += '&{v}'.format(v = var)

    response=requests.get(base)
    data_json=response.json()
    return data_json

#TA-DA!
get_data('FOD',['Tid=*','BARNKON=P'])
```

Out[33]:
```
{'dataset': {'dimension': {'BARNKON': {'label': 'sex of child',
    'category': {'index': {'P': 0}, 'label': {'P': 'Girls'}}},
   'ContentsCode': {'label': 'Indhold',
    'category': {'index': {'FOD': 0},
     'label': {'FOD': 'Live births'},
     'unit': {'FOD': {'base': 'number', 'decimals': 0}}}},
   'Tid': {'label': 'time',
    'category': {'index': {'1973': 0,
      '1974': 1,
      '1975': 2,
      '1976': 3,
      '1977': 4,
      '1978': 5,
      '1979': 6,
      '1980': 7,
      '1981': 8,
      '1982': 9,
```

```
 '1983': 10,
 '1984': 11,
 '1985': 12,
 '1986': 13,
 '1987': 14,
 '1988': 15,
 '1989': 16,
 '1990': 17,
 '1991': 18,
 '1992': 19,
 '1993': 20,
 '1994': 21,
 '1995': 22,
 '1996': 23,
 '1997': 24,
 '1998': 25,
 '1999': 26,
 '2000': 27,
 '2001': 28,
 '2002': 29,
 '2003': 30,
 '2004': 31,
 '2005': 32,
 '2006': 33,
 '2007': 34,
 '2008': 35,
 '2009': 36,
 '2010': 37,
 '2011': 38,
 '2012': 39,
 '2013': 40,
 '2014': 41,
 '2015': 42,
 '2016': 43,
 '2017': 44},
'label': {'1973': '1973',
 '1974': '1974',
 '1975': '1975',
 '1976': '1976',
 '1977': '1977',
 '1978': '1978',
 '1979': '1979',
 '1980': '1980',
 '1981': '1981',
 '1982': '1982',
 '1983': '1983',
 '1984': '1984',
 '1985': '1985',
```

```
    '1986': '1986',
    '1987': '1987',
    '1988': '1988',
    '1989': '1989',
    '1990': '1990',
    '1991': '1991',
    '1992': '1992',
    '1993': '1993',
    '1994': '1994',
    '1995': '1995',
    '1996': '1996',
    '1997': '1997',
    '1998': '1998',
    '1999': '1999',
    '2000': '2000',
    '2001': '2001',
    '2002': '2002',
    '2003': '2003',
    '2004': '2004',
    '2005': '2005',
    '2006': '2006',
    '2007': '2007',
    '2008': '2008',
    '2009': '2009',
    '2010': '2010',
    '2011': '2011',
    '2012': '2012',
    '2013': '2013',
    '2014': '2014',
    '2015': '2015',
    '2016': '2016',
    '2017': '2017'}},
 'id': ['BARNKON', 'ContentsCode', 'Tid'],
 'size': [1, 1, 45],
 'role': {'metric': ['ContentsCode'], 'time': ['Tid']}},
'label': 'Live births by sex of child, Indhold and time',
'source': 'Statistics Denmark',
'updated': '2018-02-21T07:00:00Z',
'value': [34996,
 34771,
 35260,
 31533,
 30055,
 30161,
 28909,
 27941,
 25972,
 25595,
```

```
24821,
25228,
26284,
26878,
27142,
28520,
29876,
30813,
31353,
32914,
32760,
34027,
33885,
32819,
32899,
32116,
32341,
32652,
31961,
31109,
31441,
31539,
31459,
31580,
31267,
31507,
30557,
30946,
28984,
28131,
27283,
27616,
28357,
29833,
29930]}}
```

1.3 Problems from exercise set 4

```
In [34]: import numpy as np
         import pandas as pd
```

Ex. 4.1.1: Use Pandas' CSV reader to fetch daily data weather from 1864 for various stations - available here.

Hint 1: for compressed files you may need to specify the keyword `compression`.

Hint 2: keyword `header` can be specified as the CSV has no column names.

Hint 3: Specify the path, as the URL linking directly to the 1864 file.

In [35]:
```
# [Answer to Ex. 4.1.1 here]
url = "https://www1.ncdc.noaa.gov/pub/data/ghcn/daily/by_year/1864.csv.gz"
df = pd.read_csv(url, sep=",", compression="gzip", header=None, usecols=[0,1,2,3] )
df.head(5) #The first 5 observations are examined. Looks good!
```

Out[35]:

	0	1	2	3
0	ITE00100550	18640101	TMAX	10
1	ITE00100550	18640101	TMIN	-23
2	ITE00100550	18640101	PRCP	25
3	ASN00079028	18640101	PRCP	0
4	USC00064757	18640101	PRCP	119

Ex. 4.1.2: Structure your weather DataFrame by using only the relevant columns (station identifier, data, observation type, observation value), rename them. Make sure observations are correctly formated (how many decimals should we add? one?).

Hint: rename can be done with `df.columns=COLS` where `COLS` is a list of column names.

In [36]:
```
# [Answer to Ex. 4.1.2 here]
COLS = ["station", "datetime", "obs_type", "obs_value"]
df.columns = COLS
df["obs_value"] = df["obs_value"] / 10
df.head(5)
```

Out[36]:

	station	datetime	obs_type	obs_value
0	ITE00100550	18640101	TMAX	1.0
1	ITE00100550	18640101	TMIN	-2.3
2	ITE00100550	18640101	PRCP	2.5
3	ASN00079028	18640101	PRCP	0.0
4	USC00064757	18640101	PRCP	11.9

Ex. 4.1.3: Select data for the station `ITE00100550` and only observations for maximal temperature. Make a copy of the DataFrame. Explain in a one or two sentences how copying works.

Hint 1: the `&` operator works elementwise on boolean series (like `and` in core python).

Hint 2: copying of the dataframe is done with the `copy` method for DataFrames.

In [37]:
```
# [Answer to Ex. 4.1.3 here]
dfITE = df[(df.station == "ITE00100550") & (df.obs_type == "TMAX")].copy()
dfITE.head()

#To ensure the original DataFrame remains unchanged, we use copy.
#Had we not copied the DataFrame, but simply created a new (i.e. without .copy())
#we would reference the new to the original (i.e. changes to the new would apply to t
```

Out[37]:

	station	datetime	obs_type	obs_value
0	ITE00100550	18640101	TMAX	1.0
75	ITE00100550	18640102	TMAX	0.8
152	ITE00100550	18640103	TMAX	-2.8
227	ITE00100550	18640104	TMAX	0.0
305	ITE00100550	18640105	TMAX	-1.9

Ex. 4.1.4: Make a new column called `TMAX_F` where you have converted the temperature variables to Fahrenheit.

Hint: Conversion is $F = 32 + 1.8 * C$ where F is Fahrenheit and C is Celsius.

In [38]:
```
# [Answer to Ex. 4.1.4 here]
dfITE["TMAX_F"] = 32 + 1.8 * dfITE["obs_value"]
dfITE.head()
```

Out[38]:

	station	datetime	obs_type	obs_value	TMAX_F
0	ITE00100550	18640101	TMAX	1.0	33.80
75	ITE00100550	18640102	TMAX	0.8	33.44
152	ITE00100550	18640103	TMAX	-2.8	26.96
227	ITE00100550	18640104	TMAX	0.0	32.00
305	ITE00100550	18640105	TMAX	-1.9	28.58

1.4 Problems from exercise set 5

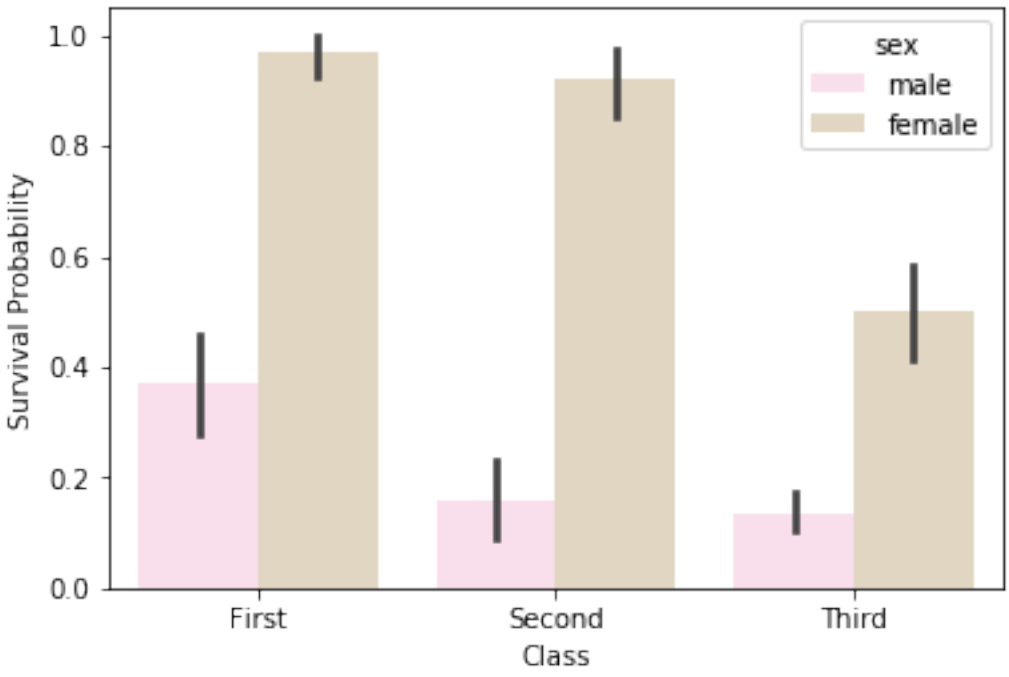
In [39]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns

%matplotlib inline

iris = sns.load_dataset('iris')
titanic = sns.load_dataset('titanic')

#For extra eye-candy effect of our seaborn plots, we set the palette.
#DISCLAIMER: If the color palette is too light for you screen to see the plots proper
    # 1) Get a better screen.
    # 2) Get a better graphic card.
    # 3) Change the color palette (this is the boring soultion ;-)
sns.set_palette("Pastel1_r")
```

Ex. 5.1.1:: Show the first five rows of the titanic dataset. What information is in the dataset? Use a barplot to show the probability of survival for men and women within each passenger class. Can you make a boxplot showing the same information (why/why not?). *Bonus:* show a boxplot for the fare-prices within each passenger class.

Spend five minutes discussing what you can learn about the survival-selection aboard titanic from the figure(s).

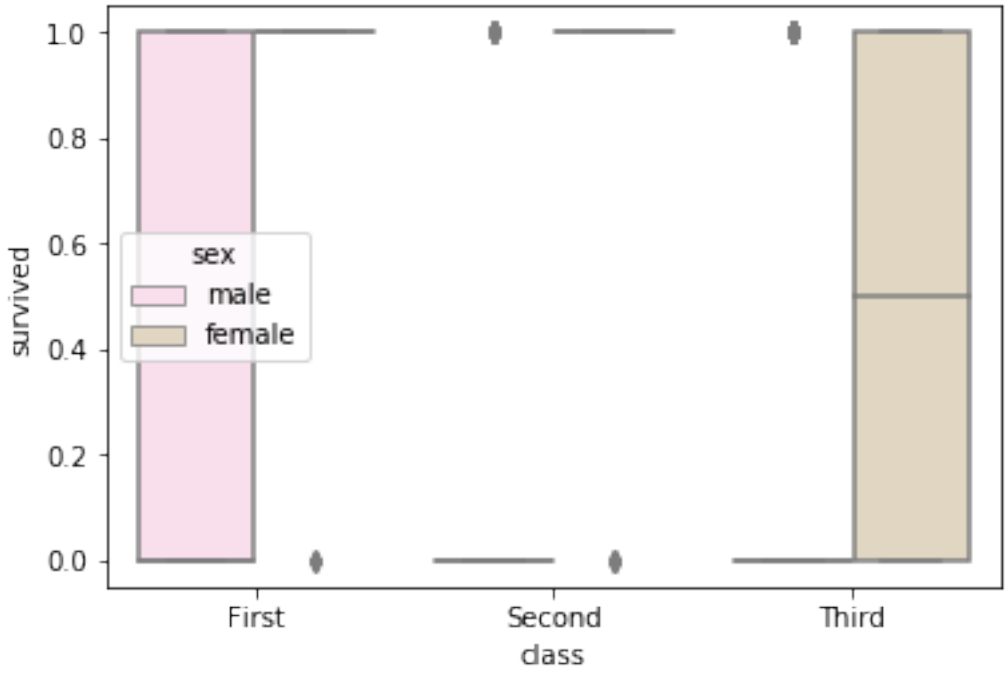
Hint: https://seaborn.pydata.org/generated/seaborn.barplot.html, specifically the `hue` option.

In [40]:
```
# [Answer to Ex. 5.1.1 here]
titanic.head(5)

#Looks like being a female on 1. and 2. class had 90-100 chance of surviving (on aver
#Whereas females on 3. class have 50% (on average) chance of surviving.
#As for the males, their odds were far worse (As the saying goes: "Save the women and
h = sns.barplot(x="class", y="survived", hue="sex", data = titanic)
plt.xlabel("Class")
plt.ylabel("Survival Probability")
plt.show(h)
```



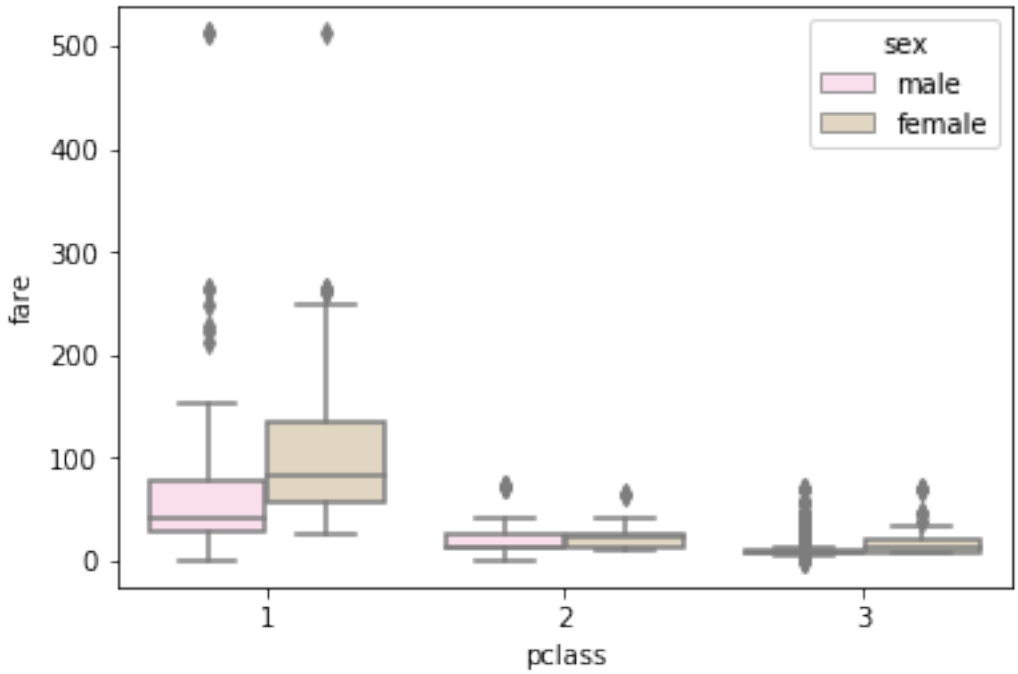
In [41]:
```
sns.boxplot(x="class", y="survived", hue="sex", data=titanic)
#Using a boxplot for a categorical/discrete variables, such as survived (either survi
#will not be informatirve nor eye candy for anyone.
```

Out[41]:
```
<matplotlib.axes._subplots.AxesSubplot at 0xd1450b8>
```



In [42]:
```
## BONUS ##
#Oh boy there's a big difference between the prices of the classes and within first-c
sns.boxplot(x="pclass", y="fare", hue="sex", data = titanic)
```

Out[42]: <matplotlib.axes._subplots.AxesSubplot at 0xd640e80>



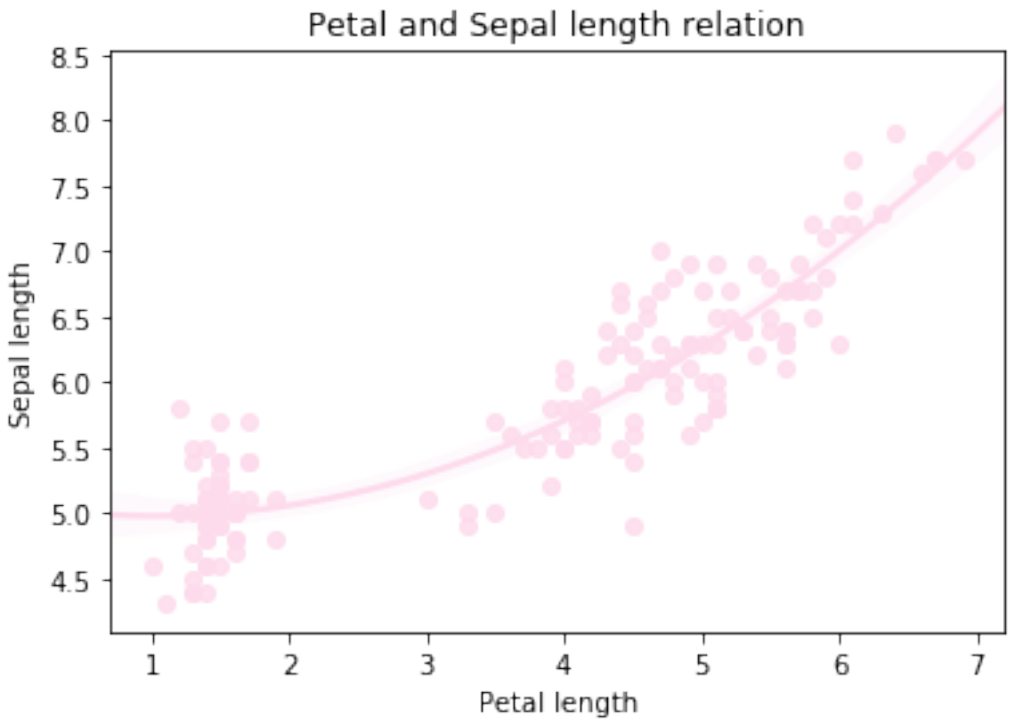
Ex. 5.1.2: Using the iris flower dataset, draw a scatterplot of sepal length and petal length. Include a second order polynomial fitted to the data. Add a title to the plot and rename the axis labels. *Discuss:* Is this a meaningful way to display the data? What could we do differently?

For a better understanding of the dataset this image might be useful:

Hint: use the `.regplot` method from seaborn.

```
In [43]: # [Answer to Ex. 5.1.2 here]

         print(iris.head(5))
         g = sns.regplot(y="sepal_length", x="petal_length", fit_reg=True, order=2, data=iris)
         plt.ylabel("Sepal length")
         plt.xlabel("Petal length")
         plt.title("Petal and Sepal length relation")
         plt.show(g)
         #Generally, using a scatterplot for this kind of data is a meaningful way to display
         #Yet, due to no flowers having having a petal length between 2-3,
         #the second order polynomial might not be the best tool.
```

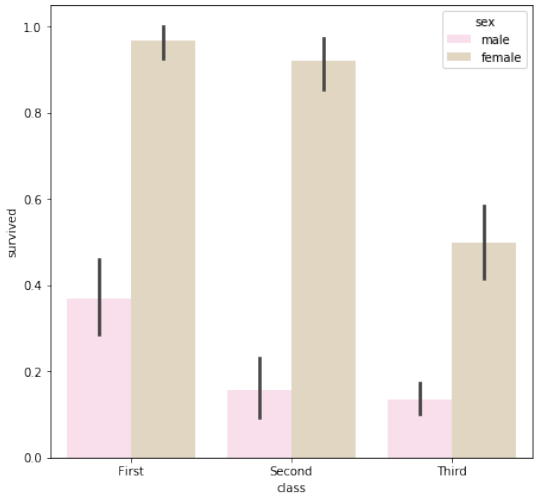
```
   sepal_length  sepal_width  petal_length  petal_width species
0           5.1          3.5           1.4          0.2  setosa
1           4.9          3.0           1.4          0.2  setosa
2           4.7          3.2           1.3          0.2  setosa
3           4.6          3.1           1.5          0.2  setosa
4           5.0          3.6           1.4          0.2  setosa
```

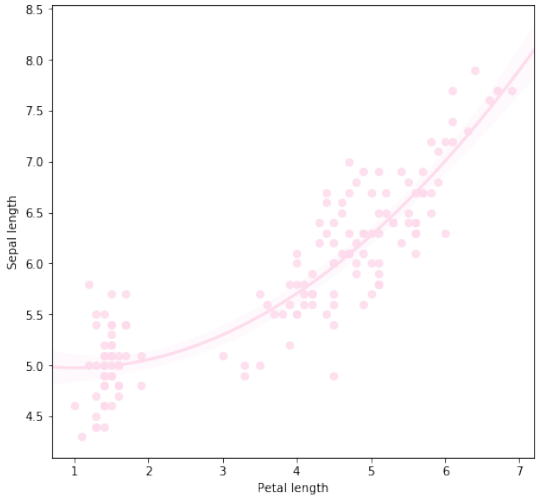


Ex. 5.1.3: Combine the two of the figures you created above into a two-panel figure similar to the one shown here:

Save the figure as a png file on your computer. > *Hint:* See this question on stackoverflow for inspiration.

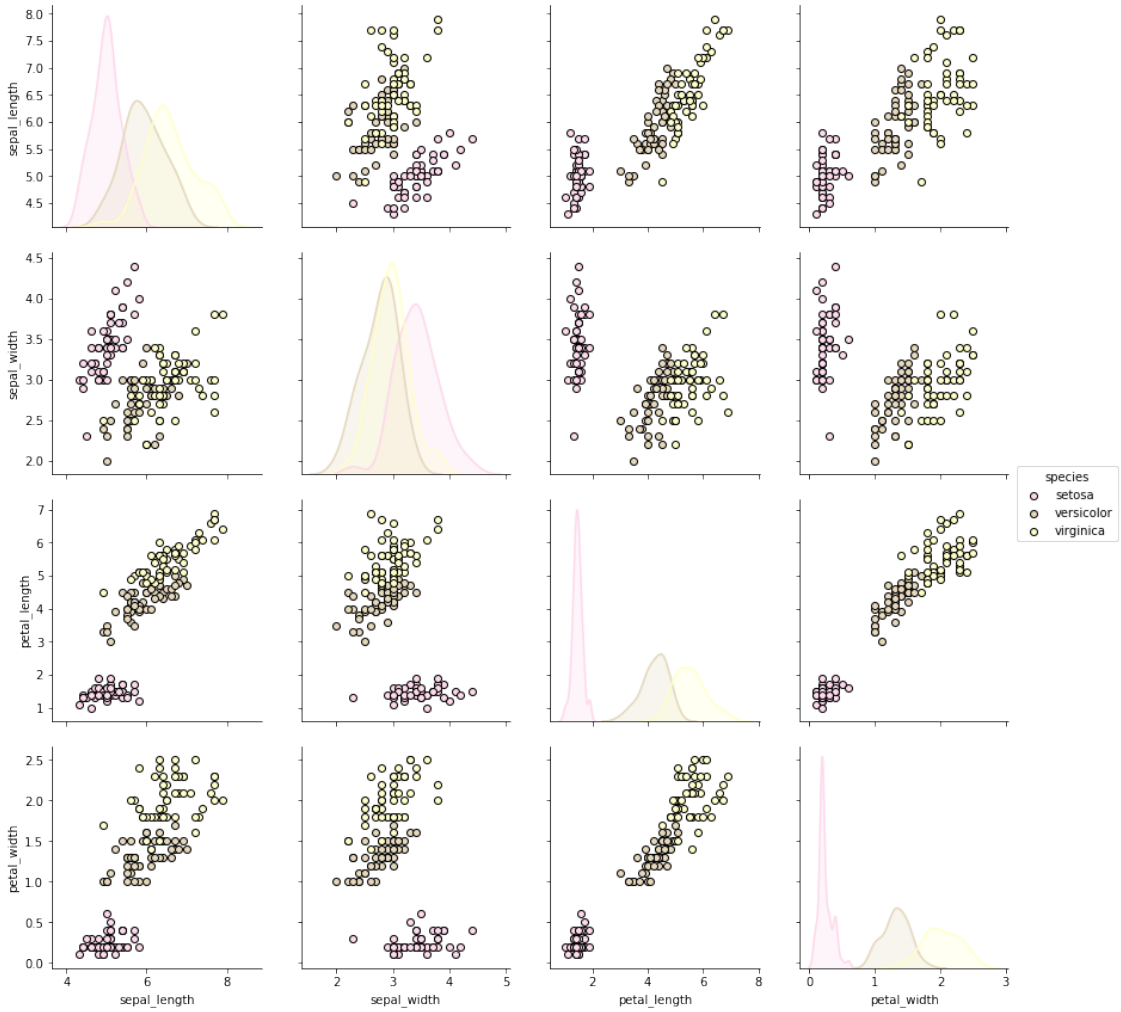
```
In [44]: # [Answer to Ex. 5.1.3 here]
         f, axes = plt.subplots(1, 2, figsize=(16,7))

         sns.barplot(x="class", y="survived", hue="sex", data = titanic, ax=axes[0])
         plt.xlabel("Class")
         plt.ylabel("Survival Probability")

         sns.regplot(y="sepal_length", x="petal_length", fit_reg=True, order=2, data=iris, ax=a
         plt.ylabel("Sepal length")
         plt.xlabel("Petal length")

         plt.savefig("Exercise 5-1-3.png")
```





Ex. 5.1.4: Use pairplot with hue to create a figure that clearly shows how the different species vary across measurements. Change the color palette and remove the shading from the density plots. *Bonus:* Try to explain how the `diag_kws` argument works (*hint:* read here)

```
In [45]: # [Answer to Ex. 5.1.4 here]
         #Coder's comment:
         #Despite the exercise explicity stating to remove the shading, we refuse to do so.
         #The density plots looks like deliciously-looking eye-candy *with* the shading
         #and as such, we refuse to remove the shading.
         #Now, one could probably argue, that we could change our colour palette to a more sui
         #but that would be no fun (and look at that beautiful colour palette!)
         sns.pairplot(iris, hue='species', size=3, kind='scatter',
                      plot_kws=dict(edgecolor="k", linewidth=1.0),
                      diag_kws=dict(shade=True), # "diag" adjusts/tunes the diagonal plots
                      diag_kind="kde")

Out[45]: <seaborn.axisgrid.PairGrid at 0xcc4edd8>
```



1.5 Problems from exercise set 6

Note: A central part of these exercises and the ones from exercise set 7 is downloading data from the NOAA servers. If you cannot complete this part, you can download the data as csv files **from github**.

```
In [46]: %matplotlib inline
         import pandas as pd
         import matplotlib.pyplot as plt
```

Ex. 6.1.4: Extract the country code from the station name into a separate column.

Hint: The station column contains a GHCND ID, given to each weather station by NOAA. The format of these ID's is a 2-3 letter country code, followed by a integer identifying the specific station. A simple approach is to assume a fixed length of the country ID. A more complex way would be to use the `re` module.

In [47]:
```
#Thanks to Kristian's beautiful code, we shall use this for transparency, instead of

url = 'https://www1.ncdc.noaa.gov/pub/data/ghcn/daily/by_year/1864.csv.gz'

df_weather = pd.read_csv(url,
                         compression='gzip',
                         header=None).iloc[:,:4]

df_weather.columns = ['station', 'datetime', 'obs_type', 'obs_value']
df_weather['obs_value'] = df_weather['obs_value'] / 10
df_select = df_weather[(df_weather.station == 'ITE00100550') & (df_weather.obs_type ==
df_select['TMAX_F'] = 32 + 1.8 * df_select['obs_value']
df_sorted = df_select.reset_index(drop=True).sort_values(by=['obs_value'])
wdf = pd.DataFrame(df_sorted)

# [Answer to Ex. 6.1.4]

import re

#Using the re module, we identify all the letters (lower and upper) for each string i
#and add them as a seperate string in a seperate column.
wdf["country"] = [''.join(re.findall("[a-zA-Z]+", item)) for item in wdf["station"]]
wdf.head(10)
```

Out[47]:

	station	datetime	obs_type	obs_value	TMAX_F	country
16	ITE00100550	18640117	TMAX	-6.3	20.66	ITE
17	ITE00100550	18640118	TMAX	-5.0	23.00	ITE
13	ITE00100550	18640114	TMAX	-5.0	23.00	ITE
12	ITE00100550	18640113	TMAX	-4.3	24.26	ITE
14	ITE00100550	18640115	TMAX	-3.1	26.42	ITE
2	ITE00100550	18640103	TMAX	-2.8	26.96	ITE
15	ITE00100550	18640116	TMAX	-2.5	27.50	ITE
11	ITE00100550	18640112	TMAX	-2.5	27.50	ITE
4	ITE00100550	18640105	TMAX	-1.9	28.58	ITE
41	ITE00100550	18640211	TMAX	-1.8	28.76	ITE

Ex. 6.1.5: Make a function that downloads and formats the weather data according to previous exercises in Exercise Section 4.1, 6.1. You should use data for ALL stations but still only select maximal temperature. *Bonus:* To validate that your function works plot the temperature curve for each country in the same window. Use `plt.legend()` to add a legend.

In [77]:
```
# [Answer to Ex. 6.1.5]
import re
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np
```

```
import pandas as pd

def get_weather_data(year_id):
    url = 'https://www1.ncdc.noaa.gov/pub/data/ghcn/daily/by_year/{id}.csv.gz'.format

    df_weather = pd.read_csv(url,
                         compression='gzip',
                         header=None).iloc[:,:4]

    df_weather.columns = ['station', 'datetime', 'obs_type', 'obs_value']
    df_weather['obs_value'] = df_weather['obs_value'] / 10
    df_select = df_weather[df_weather.obs_type == 'TMAX'].copy()
    df_select['TMAX_F'] = 32 + 1.8 * df_select['obs_value']
    df_sorted = df_select.reset_index(drop=True).sort_values(by=['obs_value'])
    df = pd.DataFrame(df_sorted)
    df['Date']=pd.to_datetime(df['datetime'],format='%Y%m%d')
    df['Month'] = pd.DatetimeIndex(df['Date']).month
    #df['Country_code']=df['station'].str[:3] #The static way of getting the country
    df["country"] = [''.join(re.findall("[a-zA-Z]+", item)) for item in df["station"]]
    return df
#The plotting is currently part of the entire function.
#Yet, the function could easily be two seperate functions,
#such that a function for fetching the data and a function for plotting exist.
```

In [78]:
```
#The results of the magnificent function  - in all its glory - can be witnessed below
get_weather_data(1864).head(10)
```

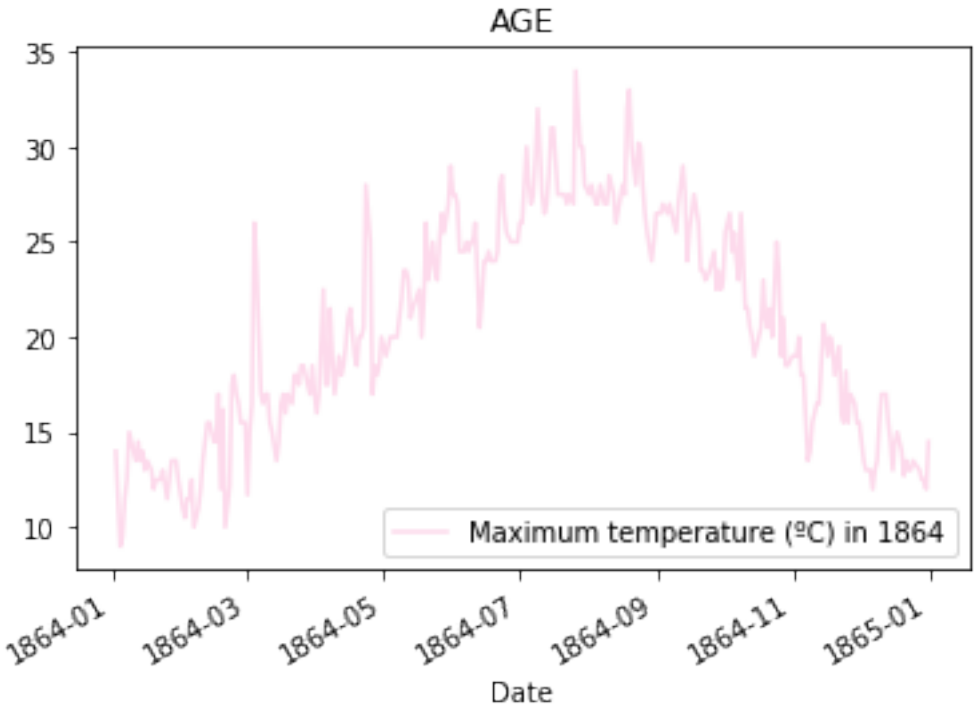
Out[78]:

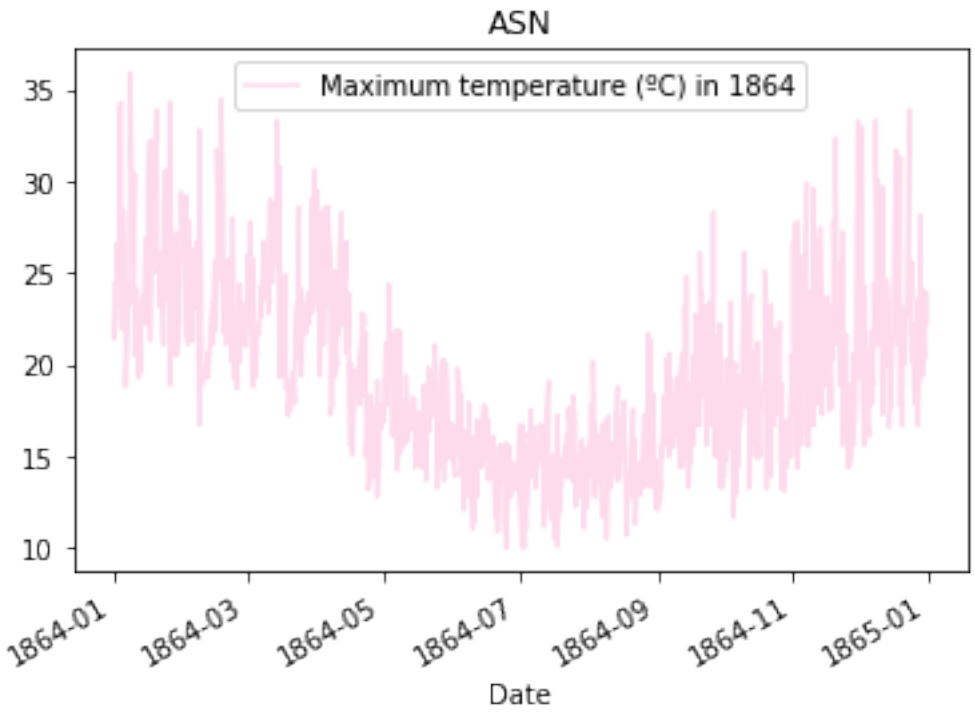
	station	datetime	obs_type	obs_value	TMAX_F	Date	Month	country
845	SZ000006717	18640226	TMAX	-34.0	-29.20	1864-02-26	2	SZ
577	SZ000006717	18640208	TMAX	-17.4	0.68	1864-02-08	2	SZ
699	CA006158350	18640217	TMAX	-16.7	1.94	1864-02-17	2	CA
42	SZ000006717	18640103	TMAX	-16.5	2.30	1864-01-03	1	SZ
16	CA006158350	18640102	TMAX	-16.1	3.02	1864-01-02	1	CA
607	SZ000006717	18640210	TMAX	-15.9	3.38	1864-02-10	2	SZ
199	EZE00100082	18640114	TMAX	-15.1	4.82	1864-01-14	1	EZE
592	SZ000006717	18640209	TMAX	-15.0	5.00	1864-02-09	2	SZ
741	SZ000006717	18640219	TMAX	-14.7	5.54	1864-02-19	2	SZ
533	SZ000006717	18640205	TMAX	-14.4	6.08	1864-02-05	2	SZ

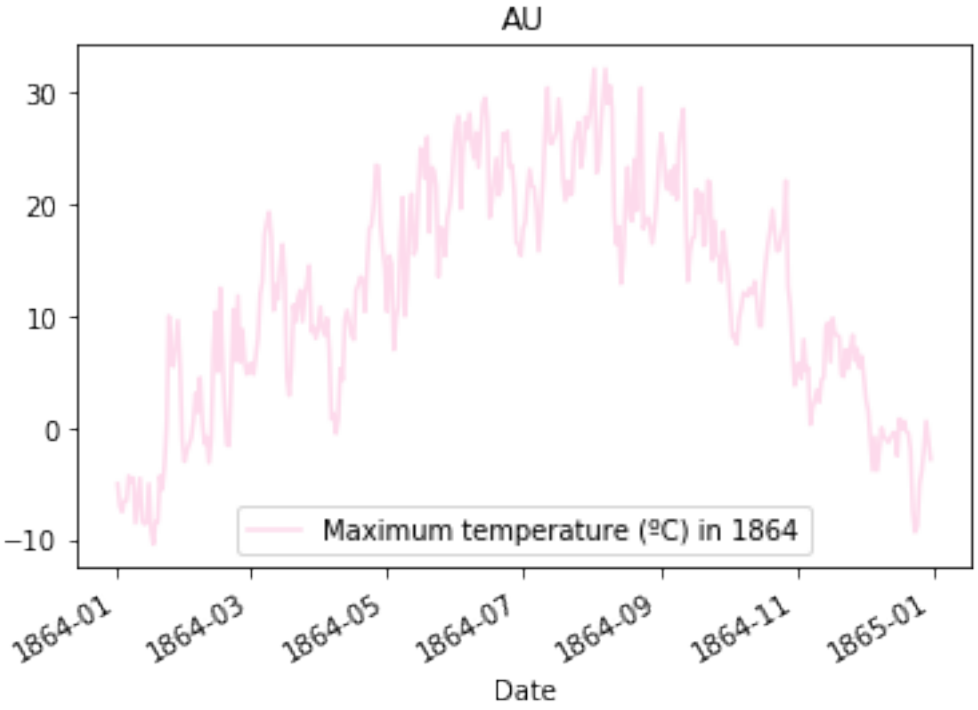
```
741      SZ
533      SZ
```

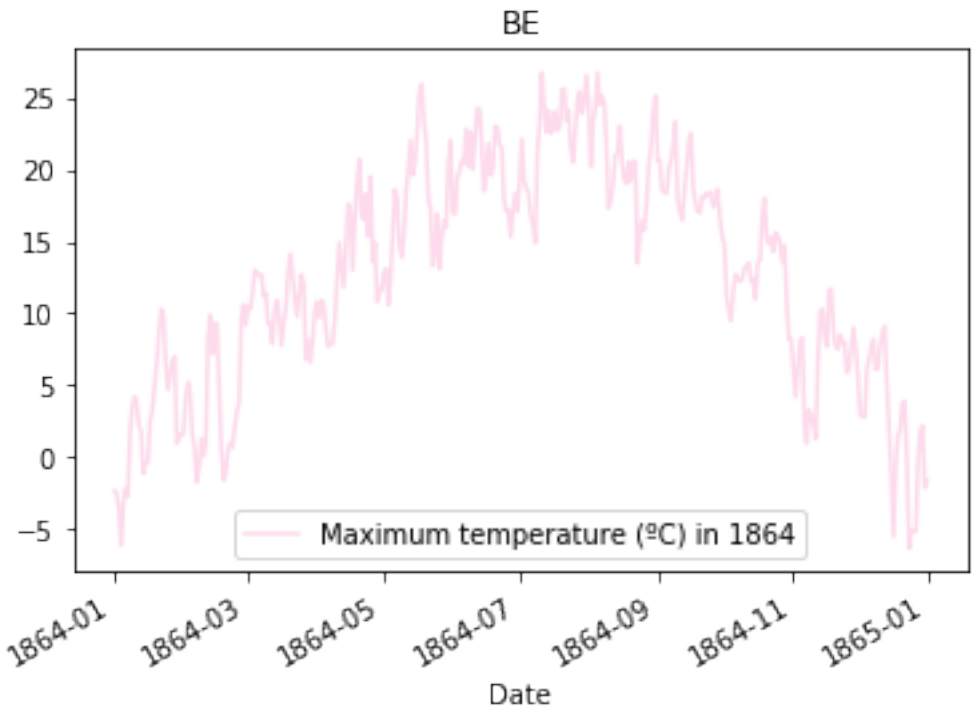
In [79]:
```
#BONUS: A FUNCTION FOR THE BEAUTIFUL PLOTS!
def plot_weather_data(year_id):
    for title, group in get_weather_data(year_id).groupby("country"):
        group.plot(x='Date', y='obs_value', title=title)
        plt.legend(['Maximum temperature (žC) in ' + str(year_id)])
```

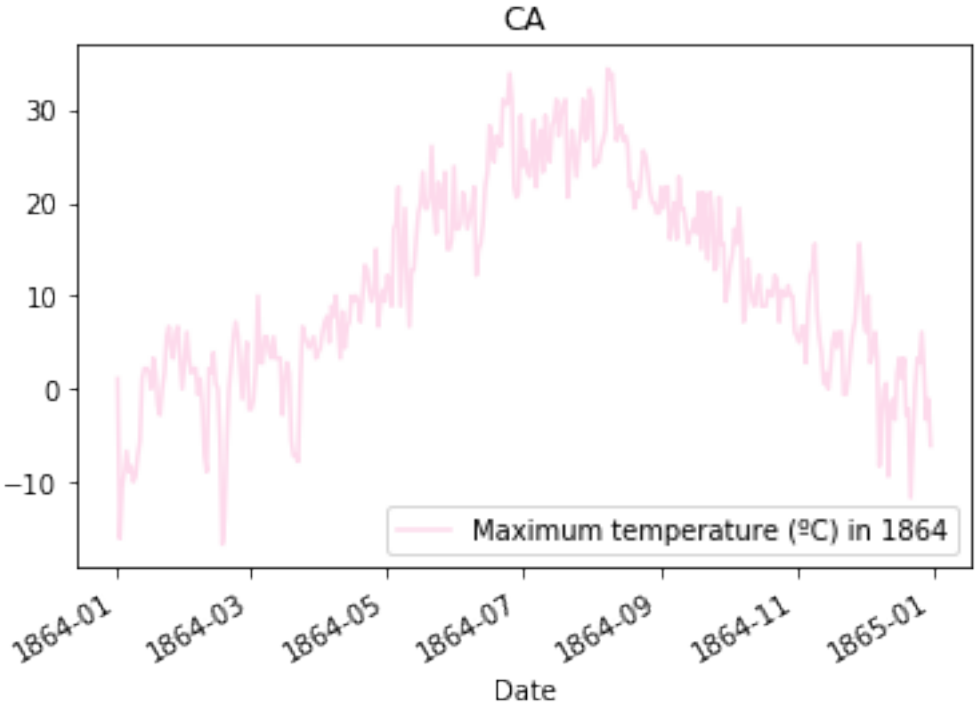
In [80]:
```
plot_weather_data(1864)
```

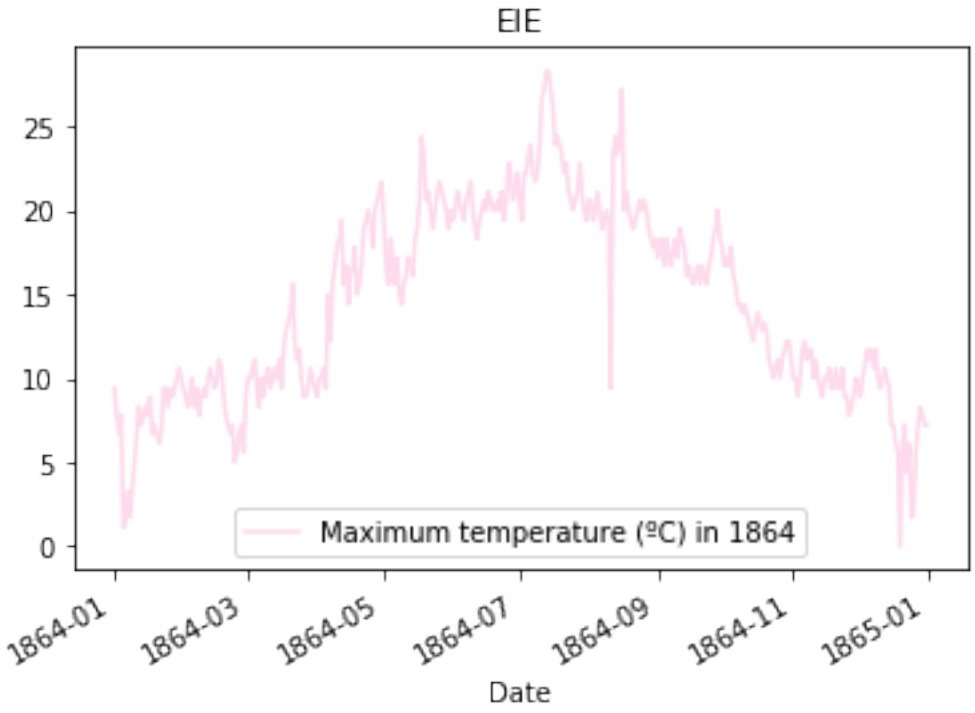


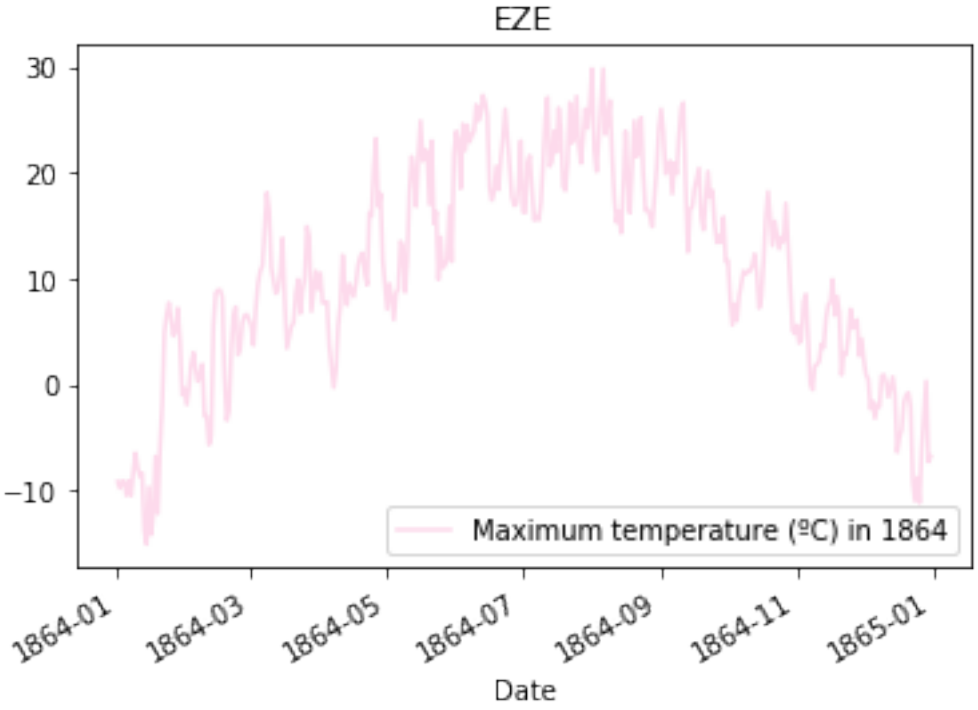

ASN
Maximum temperature (ºC) in 1864
35
30
25
20
15
10
1864-01
1864-03
1864-05
1864-07
1864-09
1864-11
1865-01
Date

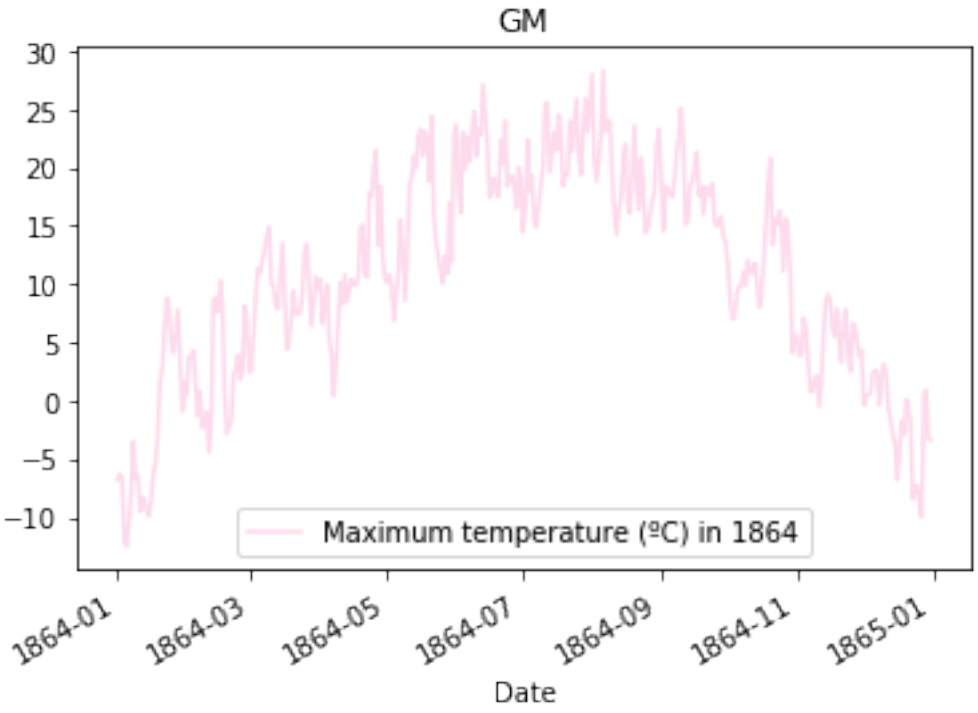

AU
30
20
10
0
−10
Maximum temperature (ºC) in 1864
1864-01
1864-03
1864-05
1864-07
1864-09
1864-11
1865-01
Date

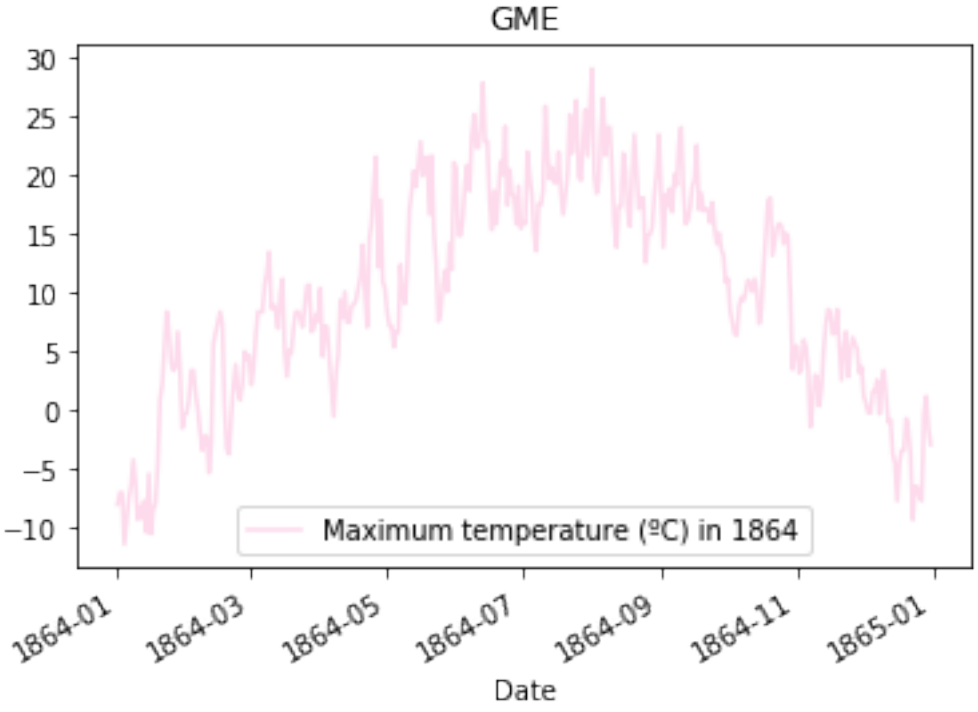

BE
25
20
15
10
5
0
−5
Maximum temperature (ºC) in 1864
1864-01
1864-03
1864-05
1864-07
1864-09
1864-11
1865-01
Date

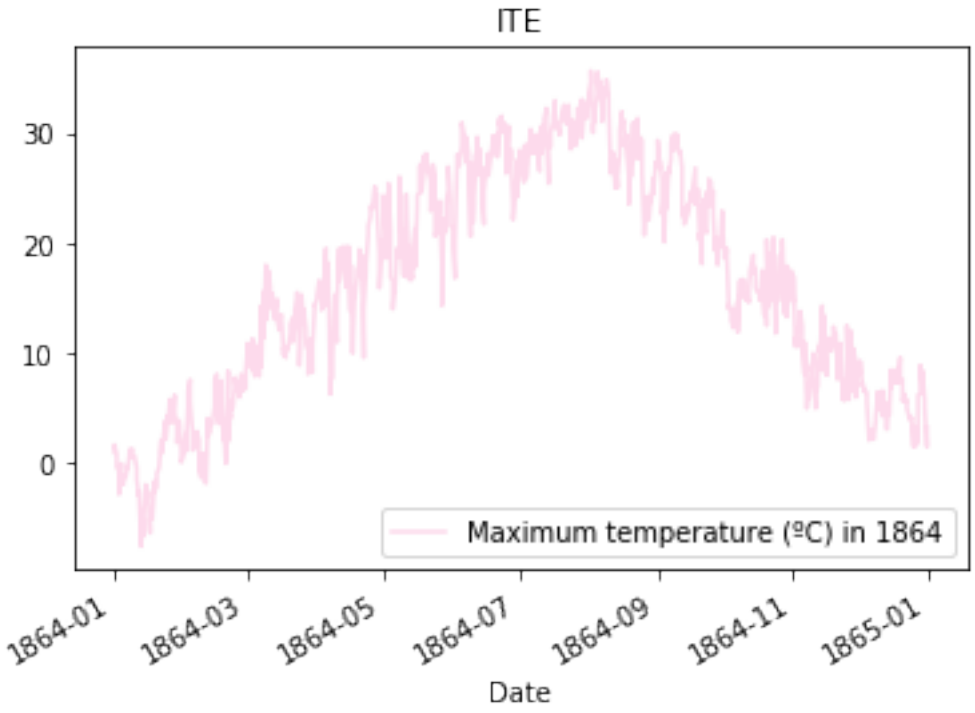

CA
30
20
10
0
−10
Maximum temperature (ºC) in 1864
1864-01
1864-03
1864-05
1864-07
1864-09
1864-11
1865-01
Date

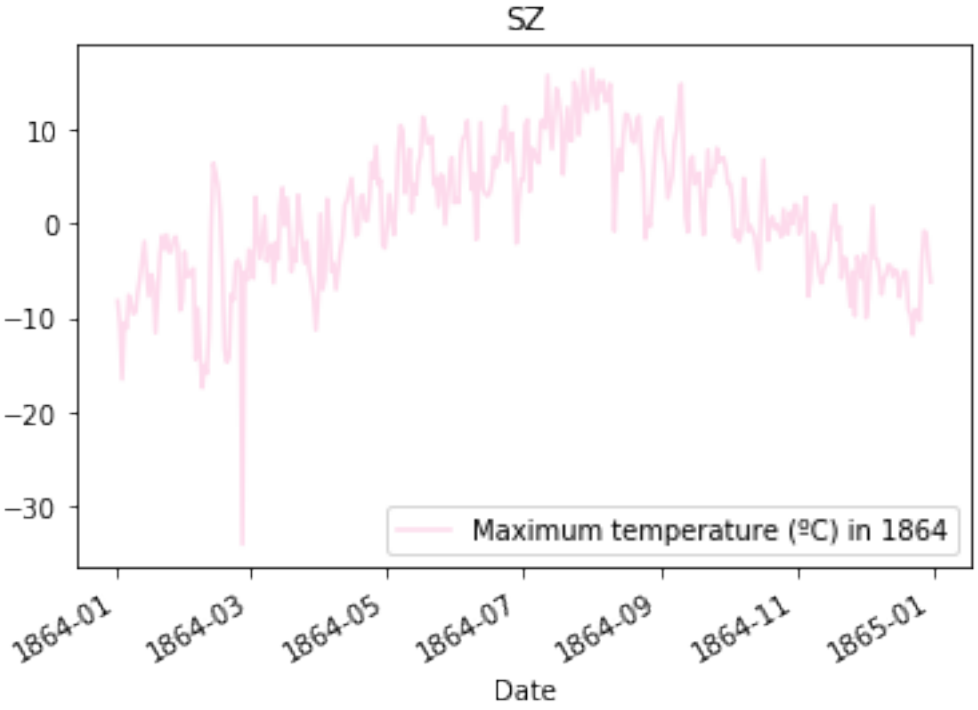

EIE
25
20
15
10
5
0
Maximum temperature (ºC) in 1864
1864-01
1864-03
1864-05
1864-07
1864-09
1864-11
1865-01
Date

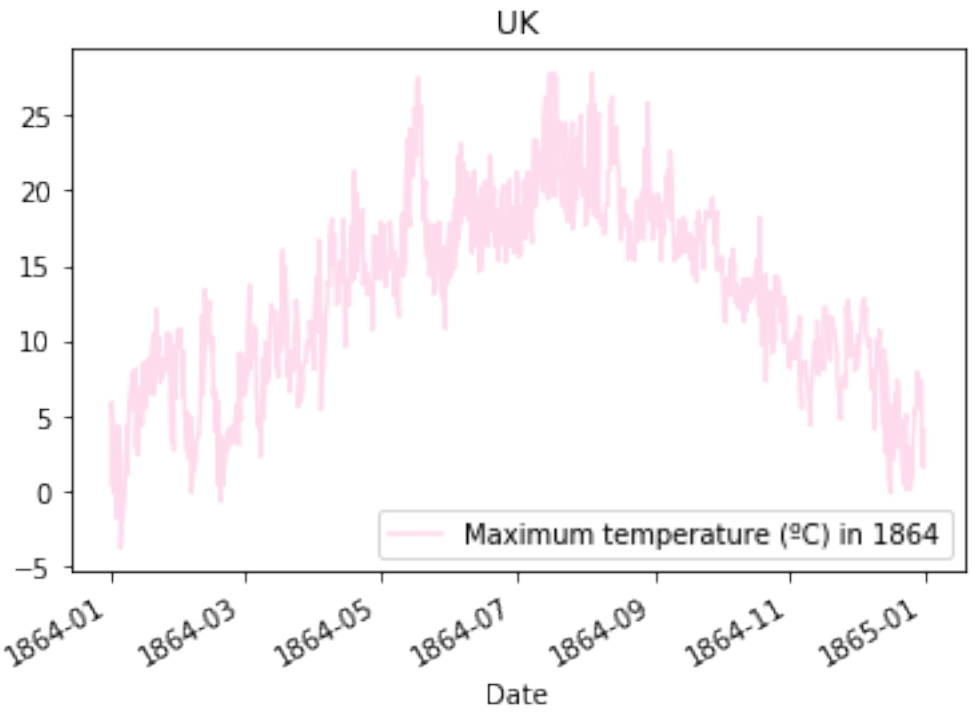

EZE
30
20
10
0
−10
Maximum temperature (ºC) in 1864
1864-01
1864-03
1864-05
1864-07
1864-09
1864-11
1865-01
Date


GM
30
25
20
15
10
5
0
−5
−10
Maximum temperature (ºC) in 1864
1864-01
1864-03
1864-05
1864-07
1864-09
1864-11
1865-01
Date


GME
30
25
20
15
10
5
0
−5
−10
Maximum temperature (ºC) in 1864
1864-01
1864-03
1864-05
1864-07
1864-09
1864-11
1865-01
Date


ITE
30
20
10
0
Maximum temperature (ºC) in 1864
1864-01
1864-03
1864-05
1864-07
1864-09
1864-11
1865-01
Date


SZ
10
0
−10
−20
−30
Maximum temperature (ºC) in 1864
1864-01
1864-03
1864-05
1864-07
1864-09
1864-11
1865-01
Date



1.6 Problems from exercise set 7

Note: Once again if you haven't managed to download the data from NOAA, you can refer to the github repo to get csv-files containing the required data.

```
In [81]: %matplotlib inline

         import pandas as pd
         import matplotlib.pyplot as plt
         import matplotlib as mpl

         # Increases the plot size a little
         mpl.rcParams['figure.figsize'] = 22, 12
```

Ex. 7.1.1: Plot the monthly max,min, mean, first and third quartiles for maximum temperature for our station with the ID *'ITE00100550'* in 1864.

Hint: the method `describe` computes all these measures.

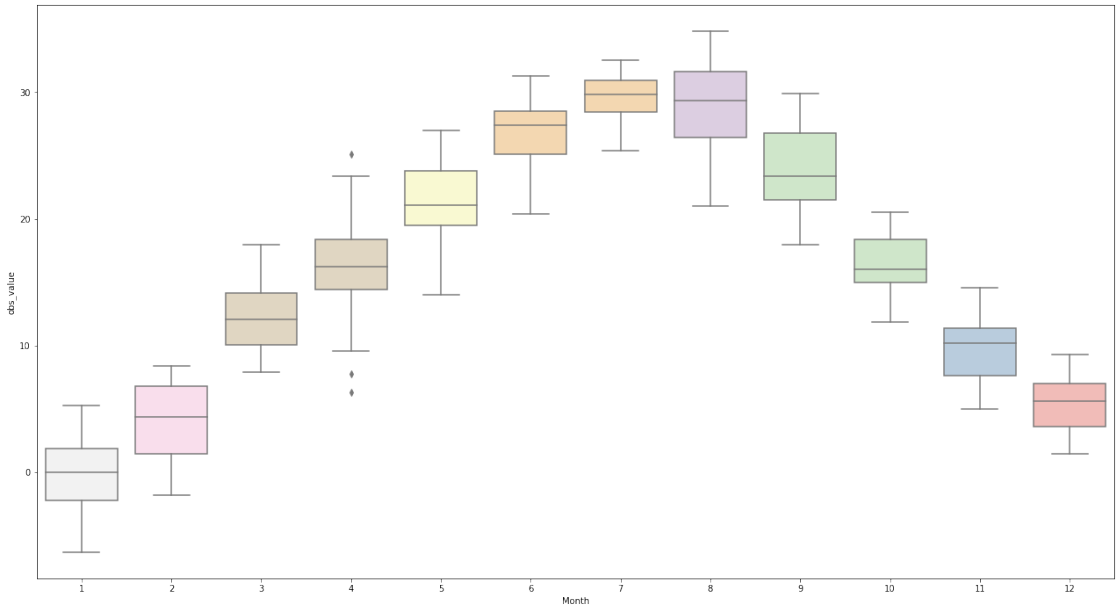
```
In [93]: # [Answer to Ex. 7.1.1]
         df1 = get_weather_data(1864) #Data is retreived based on our previous function.
         df_ITE =  df1[(df1.station == 'ITE00100550')].copy() #A sub-sample for the desired st
         df2 = df_ITE[["obs_value", "TMAX_F", "Month"]].copy() #A sub-sample is created for ea
         df2.describe() #No need to look at summary statistics for datetime :)
```

Out[93]:

	obs_value	TMAX_F	Month
count	366.000000	366.000000	366.000000
mean	16.249727	61.249508	6.513661
std	10.158772	18.285789	3.455958
min	-6.300000	20.660000	1.000000
25%	7.600000	45.680000	4.000000
50%	16.150000	61.070000	7.000000
75%	25.325000	77.585000	9.750000
max	34.800000	94.640000	12.000000

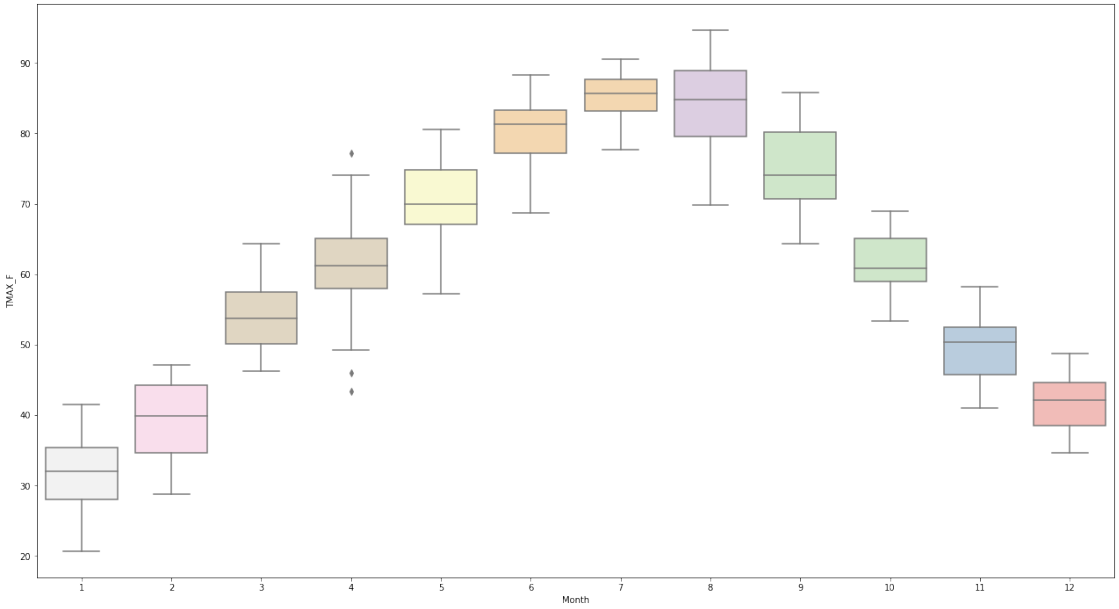
In [95]:
```
#In Celsius:
sns.boxplot(y = "obs_value", x= "Month", data=df2, palette="Pastel1_r")
```

Out[95]: <matplotlib.axes._subplots.AxesSubplot at 0xd351908>



In [96]:
```
#In Fahrenheir:
sns.boxplot(y = "TMAX_F", x= "Month", data=df2, palette="Pastel1_r")
```

Out[96]: <matplotlib.axes._subplots.AxesSubplot at 0xd0cd208>



Ex. 7.1.2: Get the processed data from years 1864-1867 as a list of DataFrames. Convert the list into a single DataFrame by concatenating vertically.

```
In [97]: # [Answer to Ex. 7.1.2]
         #The same function as made in exercise 6 is used:
         #A function, based on the original code for fetching the data, where a year_id is spe
         #As such, the processed data from whichever desire year can be retrieved by specifyin
         def get_weather_data(year_id):
             url = 'https://www1.ncdc.noaa.gov/pub/data/ghcn/daily/by_year/{id}.csv.gz'.format

             df_weather = pd.read_csv(url,
                                      compression='gzip',
                                      header=None).iloc[:,:4]

             df_weather.columns = ['station', 'datetime', 'obs_type', 'obs_value']
             df_weather['obs_value'] = df_weather['obs_value'] / 10
             df_select = df_weather[df_weather.obs_type == 'TMAX'].copy()
             df_select['TMAX_F'] = 32 + 1.8 * df_select['obs_value']
             df_sorted = df_select.reset_index(drop=True).sort_values(by=['obs_value'])
             df = pd.DataFrame(df_sorted)
             df['Date']=pd.to_datetime(df['datetime'],format='%Y%m%d')
             df['Month'] = pd.DatetimeIndex(df['Date']).month
             #df['Country_code']=df['station'].str[:3] #The static way of getting the country
             df["country"] = [''.join(re.findall("[a-zA-Z]+", item)) for item in df["station"]]
             return df

In [101]: yearlist = []
          for i in range(1864,1868):
```

```
        yearlist.append(get_weather_data(i))

    df4 = pd.concat(yearlist)

    #A more manual way can be seen below, yet looping is a lot simpler and smarter for l
        # wdf4 = pd.concat([get_weather_data(1864), get_weather_data(1865), get_weather_
```

```
In [123]: df4.head(10)
```

Out[123]:

	station	datetime	obs_type	obs_value	TMAX_F	Date	Month	\
845	SZ000006717	18640226	TMAX	-34.0	-29.20	1864-02-26	2	
577	SZ000006717	18640208	TMAX	-17.4	0.68	1864-02-08	2	
699	CA006158350	18640217	TMAX	-16.7	1.94	1864-02-17	2	
42	SZ000006717	18640103	TMAX	-16.5	2.30	1864-01-03	1	
16	CA006158350	18640102	TMAX	-16.1	3.02	1864-01-02	1	
607	SZ000006717	18640210	TMAX	-15.9	3.38	1864-02-10	2	
199	EZE00100082	18640114	TMAX	-15.1	4.82	1864-01-14	1	
592	SZ000006717	18640209	TMAX	-15.0	5.00	1864-02-09	2	
741	SZ000006717	18640219	TMAX	-14.7	5.54	1864-02-19	2	
533	SZ000006717	18640205	TMAX	-14.4	6.08	1864-02-05	2	

	country
845	SZ
577	SZ
699	CA
42	SZ
16	CA
607	SZ
199	EZE
592	SZ
741	SZ
533	SZ

Ex. 7.1.3: Parse the station location data which you can find at https://www1.ncdc.noaa.gov/pub/data/ghcn/daily/ghcnd-stations.txt. Merge station locations onto the weather data spanning 1864-1867.

Hint: The location data have the folllowing format,

```
------------------------------
Variable   Columns   Type
------------------------------
ID            1-11   Character
LATITUDE     13-20   Real
LONGITUDE    22-30   Real
ELEVATION    32-37   Real
STATE        39-40   Character
NAME         42-71   Character
GSN FLAG     73-75   Character
```

```
HCN/CRN FLAG 77-79   Character
WMO ID       81-85   Character
------------------------------
```

Hint: The station information has fixed width format - does there exist a pandas reader for that?

```
In [120]: # [Answer to Ex. 7.1.3]
          #Location data is fetched and cleaned based on the its fixed width format:
          url_loc = "https://www1.ncdc.noaa.gov/pub/data/ghcn/daily/ghcnd-stations.txt"
          colspecs = [(0, 11), (12, 21), (21, 31), (31, 38), (38, 41), (41, 72), (72, 76), (76
          df_loc = pd.read_fwf(url_loc, header=None, colspecs = colspecs)
          df_loc.columns = ["station", "latitude","longitude","elevation","state","name","gsn"
          df_loc.head() #Having a look is always nice (especially when your code works)
```

```
Out[120]:        station  latitude  longitude  elevation state                    name  \
          0  ACW00011604   17.1167   -61.7833       10.1   NaN  ST JOHNS COOLIDGE FLD
          1  ACW00011647   17.1333   -61.7833       19.2   NaN               ST JOHNS
          2  AE000041196   25.3330    55.5170       34.0   NaN    SHARJAH INTER. AIRP
          3  AEM00041194   25.2550    55.3640       10.4   NaN             DUBAI INTL
          4  AEM00041217   24.4330    54.6510       26.8   NaN         ABU DHABI INTL

             gsn  hcn      wmo
          0  NaN  NaN      NaN
          1  NaN  NaN      NaN
          2  GSN  NaN  41196.0
          3  NaN  NaN  41194.0
          4  NaN  NaN  41217.0
```

```
In [122]: wdf_merged = pd.merge(wdf4, df_loc, how='left') #the two DataFrames are merged
          wdf_merged.head() #Wow such beauty, many data, very neat.
```

```
Out[122]:        station  datetime obs_type  obs_value  TMAX_F       Date  Month  \
          0  SZ000006717  18640226     TMAX      -34.0  -29.20 1864-02-26      2
          1  SZ000006717  18640208     TMAX      -17.4    0.68 1864-02-08      2
          2  CA006158350  18640217     TMAX      -16.7    1.94 1864-02-17      2
          3  SZ000006717  18640103     TMAX      -16.5    2.30 1864-01-03      1
          4  CA006158350  18640102     TMAX      -16.1    3.02 1864-01-02      1

            country  latitude  longitude  elevation state                    name  gsn  \
          0      SZ   45.8667     7.1667     2472.0   NaN  COL DU GRAND ST-BERNARD  GSN
          1      SZ   45.8667     7.1667     2472.0   NaN  COL DU GRAND ST-BERNARD  GSN
          2      CA   43.6667   -79.4000      113.0    ON                 TORONTO  NaN
          3      SZ   45.8667     7.1667     2472.0   NaN  COL DU GRAND ST-BERNARD  GSN
          4      CA   43.6667   -79.4000      113.0    ON                 TORONTO  NaN

             hcn      wmo
          0  NaN   6717.0
          1  NaN   6717.0
```

```
2  NaN  71266.0
3  NaN   6717.0
4  NaN  71266.0
```

1.7 Problems from exercise set 8

Ex. 8.1.2.: Use the `request` module to collect the first page of job postings and unpack the relevant `json` data into a `pandas` DataFrame.

```
In [125]: import requests
          import pandas as pd
          import matplotlib.pyplot as plt
```

```
In [127]: # [Answer to Ex. 8.1.2 here]
          url = "https://job.jobnet.dk/CV/FindWork/Search"
          jobdata  = requests.get(url).json()
          # response.ok
          jobdata.keys()
              #The data consists of 4 dictionairies:
                  #Expression
                  #Facets
                  #JobPositionPostings
                  #TotalResultCount
```

```
Out[127]: dict_keys(['Expression', 'Facets', 'JobPositionPostings', 'TotalResultCount'])
```

Ex. 8.1.3.: Store and print the 'TotalResultCount' value for later use. Also create a dataframe from the 'JobPositionPostings' field in the json.

```
In [128]: # [Answer to Ex. 8.1.3 here]
          TRC = jobdata["TotalResultCount"]
          print(TRC)

          df = pd.DataFrame(jobdata["JobPositionPostings"])
          df #We see the first page of the job postings. Notice the weird formated AssignmentS
```

```
15684
```

```
Out[128]:     AnonymousEmployer  AssignmentStartDate  AutomatchType  Country  \
          0               False  0001-01-01T00:00:00              0  Danmark
          1               False  0001-01-01T00:00:00              0  Danmark
          2               False  0001-01-01T00:00:00              0  Danmark
          3               False  0001-01-01T00:00:00              0  Danmark
          4               False  0001-01-01T00:00:00              0  Danmark
          5               False  0001-01-01T00:00:00              0  Danmark
          6               False  0001-01-01T00:00:00              0  Danmark
          7               False  0001-01-01T00:00:00              0  Danmark
          8               False  0001-01-01T00:00:00              0  Danmark
```

```
9              False  0001-01-01T00:00:00             0  Danmark
10             False  0001-01-01T00:00:00             0  Danmark
11             False  0001-01-01T00:00:00             0  Danmark
12             False  0001-01-01T00:00:00             0  Danmark
13             False  0001-01-01T00:00:00             0  Danmark
14             False  0001-01-01T00:00:00             0  Danmark
15             False  0001-01-01T00:00:00             0  Danmark
16             False  0001-01-01T00:00:00             0  Danmark
17             False  0001-01-01T00:00:00             0  Danmark
18             False  0001-01-01T00:00:00             0  Danmark
19             False  0001-01-01T00:00:00             0  Danmark

   EmploymentType  HasLocationValues  HiringOrgCVR  \
0                             True      35395806
1                             True      35087273
2                             True      10354331
3                             True      29201625
4                             True      29189420
5                             True      38380303
6                             True      37965766
7                             True      37965766
8                             True      37965766
9                             True      37965766
10                            True      25483863
11                            True      60729018
12                            True      29189595
13                            True      29188947
14                            True      29188947
15                            True      65307316
16                            True      19687236
17                            True      26059763
18                            True      65307316
19                            True      65307316

                       HiringOrgName       ID  IsExternal     ...      \
0            BRUNO A SØRENSEN BYG A/S  4825162       False     ...
1                    Frøken Morgenhår  4866223       False     ...
2            FIRST HOTEL ATLANTIC A/S  4866222       False     ...
3                       FLORAS CAFE    4866221       False     ...
4                      Højvangskolen   4866220       False     ...
5                    danbolig Allerød  4866219       False     ...
6   Trin for Trin Service Erhverv ApS  4866215       False     ...
7   Trin for Trin Service Erhverv ApS  4866214       False     ...
8   Trin for Trin Service Erhverv ApS  4866213       False     ...
9   Trin for Trin Service Erhverv ApS  4866212       False     ...
10            Ole Larsen Transport AS  4866208       False     ...
11       Vejdirektoratet - København   4866207       False     ...
12  Dagplejen Børnehusene ved Fjorden  4866206       False     ...
```

```
13                  Rådhuset Otterup  4866205       False     ...
14                  Rådhuset Otterup  4866204       False     ...
15                 Rødovre Jobcenter  4866202       False     ...
16                  JKS a/s, Næstved  4866201       False     ...
17                           Naviair  4866200       False     ...
18                      Islev Skole  4866198       False     ...
19           Den Kommunale Tandpleje  4866197       False     ...

    UseWorkPlaceAddressForJoblog Weight WorkHours WorkPlaceAbroad  \
0                           True    1.0   Fuldtid           False
1                           True    1.0   Fuldtid           False
2                           True    1.0   Fuldtid           False
3                           True    1.0    Deltid           False
4                           True    1.0    Deltid           False
5                           True    1.0   Fuldtid           False
6                          False    1.0    Deltid           False
7                           True    1.0    Deltid           False
8                          False    1.0    Deltid           False
9                          False    1.0    Deltid           False
10                          True    1.0   Fuldtid           False
11                          True    1.0   Fuldtid           False
12                          True    1.0   Fuldtid           False
13                          True    1.0    Deltid           False
14                          True    1.0   Fuldtid           False
15                          True    1.0   Fuldtid           False
16                          True    1.0   Fuldtid           False
17                          True    1.0   Fuldtid           False
18                          True    1.0   Fuldtid           False
19                          True    1.0    Deltid           False

         WorkPlaceAddress WorkPlaceCity  WorkPlaceNotStatic  \
0         Sønderbrogade 61       Tørring               False
1         Stationstorvet 4      Ølstykke               False
2           Europaplads 10      Aarhus C               False
3         Vesterbrogade 16   København V               False
4            Tingstedet 20   Svenstrup J               False
5   Allerød Stationsvej 2E       Allerød               False
6                                                      False
7             Savværket 32          Gram               False
8                                                      False
9                                                      False
10          Dalgårdsvej 7       Brabrand               False
11    Niels Juels Gade 13    København K               False
12              Nyrupvej 78    Kalundborg               False
13         Rådhuspladsen 2       Otterup               False
14         Rådhuspladsen 2       Otterup               False
15          Egegårdsvej 66       Rødovre               False
16        Transportbuen 5        Næstved               False
```

```
17         Naviair Alle 1        Kastrup               False
18          Islevbrovej 44       Rødovre               False
19          Rødovrevej 125       Rødovre               False

   WorkPlaceOtherAddress  WorkPlacePostalCode WorkplaceID
0                  False                 7160      121826
1                  False                 3650      108053
2                  False                 8000       14299
3                  False                 1620       55088
4                  False                 9230       95648
5                  False                 3450           0
6                   True                            119608
7                  False                 6510      119608
8                   True                            119608
9                   True                            119608
10                 False                 8220       78547
11                 False                 1059       28563
12                 False                 4400      128235
13                 False                 5450        1311
14                 False                 5450        1311
15                 False                 2610       86389
16                 False                 4700       94997
17                 False                 2770       23421
18                 False                 2610      111261
19                 False                 2610      126655

[20 rows x 42 columns]
```

1.8 Problems from exercise set 9

Ex. 9.2.1: Load the data used in the exercise using the `pd.read_csv` function. (Hint: path to file can be both a url or systempath).

Define a variable `sample_string = '\n'.join(df.sample(2000).reviewBody)` as sample of all the reviews that you will practice on. (Run it once in a while to get a new sample for potential differences). Imagine we were a company wanting to find the reviews where customers are concerned with the price of a service. They decide to write a regular expression to match all reviews where a currencies and an amount is mentioned.

```
In [129]: #Setting up:
          import pandas as pd
          import requests
          import re
          url = 'https://raw.githubusercontent.com/snorreralund/explore_regex/master/explore_re
          response = requests.get(url)
          with open('explore_regex.py','w') as f:
              f.write(response.text)
          import explore_regex as e_re
```

```
In [130]: def explore_regex(pattern,string_sample,n_output=30,before=10,after=10,shuffle=False
              count = 0
              import random
              length = len(string_sample)

              results = list(enumerate(re.finditer(pattern,string_sample)))
              re
              if shuffle:
                  random.shuffle(results)
              indices,results = zip(*results)
              for result in results:
                  start,stop = result.span()
                  temp_after = min([length-stop,after])
                  temp_before = min([start,before])
                  print('Matched: %s\tContext:%s'%(result.group(),string_sample[start-temp_befo
                  count+=1
                  if count>n_output:
                      break
              return [result.group() for result in results]

In [131]: # [Answer to Ex. 9.2.1]
          url = 'https://raw.githubusercontent.com/snorreralund/scraping_seminar/master/englis
          data = pd.read_csv(url)
          sample_string = '\n'.join(data.sample(2000).reviewBody)


          any_amount = '(\d+(?:\.\d{2})?(?:\,\d{3})?)'

          print(explore_regex(any_amount, sample_string))
```

```
Matched: 400.00        Context:efund the 400.00 we are ou
Matched: 400.00        Context:told that 400.00 isn't a l
Matched: 8        Context:ipped and 8-10 days l
Matched: 10        Context:ped and 8-10 days late
Matched: 8        Context:ould take 8 working d
Matched: 10        Context:o arrive. 10 working d
Matched: 2        Context:h.  Round 2 blew us a
Matched: 5        Context:ednesday, 5 days befo
Matched: 10        Context:y. That's 10 days???
Matched: 3        Context:arted out 3 hrs away
Matched: 6        Context:ys it was 6 hrs away.
Matched: 4        Context:der four (4) boxes? o
Matched: 2        Context:rder two (2) or more
Matched: 4        Context:ing four (4) boxes? o
Matched: 2        Context:ar as I'm 2 boxes int
Matched: 7100        Context:
Carried d7100 and 24-70
Matched: 24        Context:d7100 and 24-70 lens o
```

```
Matched: 70         Context:00 and 24-70 lens on t
Matched: 10         Context:te them a 10 out of 10
Matched: 10         Context:10 out of 10. Loved it
Matched: 12         Context:for about 12 years now
Matched: 3         Context:his is my 3rd time or
Matched: 1         Context:nately my 1st stethos
Matched: 3         Context:ght take  3-5 minutes
Matched: 5         Context:t take  3-5 minutes,
Matched: 20         Context:m saving $20 each mont
Matched: 7         Context: the card 7 days ago.
Matched: 12         Context:s.
Bought 12 Games Bun
Matched: 30         Context:h...about 30 minutes t
Matched: 50         Context:tes to do 50.
Every ti
Matched: 1         Context:.and only 1/4 of the
['400.00', '400.00', '8', '10', '8', '10', '2', '5', '10', '3', '6', '4', '2', '4', '2', '7100
```

Ex. 9.2.2: Write an expression that matches both the dollar-sign ($) and dollar written literally, and the amount before or after a dollar-sign. Remember that the "$"-sign is a special character in regular expressions. Explore and refine using the explore_pattern function in the package I created called explore_regex.

```
import explore_regex as e_re
explore_regex = e_re.Explore_Regex(sample_string) # Initaizlie the Explore regex Class.
explore_regex.explore_pattern(pattern) # Use the .explore_pattern method.
```

Start with exploring the context around digits ("") in the data.

```
In [132]: # [Answer to Ex. 9.2.2]
          part1 = r'([$]\d+(?:\.\d{2})?(?:\,\d{3})?)'
          part2 = '[0-9]+(?:[,.][0-9]+)?\s{0,5}dollar(?:oner)?'

          dollars = [part1, part2]
          generic_re = re.compile("(%s|%s)" % (part1, part2))

          print(explore_regex(generic_re,sample_string))
```

```
Matched: $20         Context:am saving $20 each mont
Matched: $49         Context:nly costs $49 bucks!
I
Matched: $2000         Context:ent nearly$2000 in my fir
Matched: $5         Context:I found a $5 I had and
Matched: 10dollar         Context: belts at 10dollarmall for p
Matched: $300         Context:dditional $300 on top of
Matched: $8,550         Context:op of the $8,550 I already
Matched: $196.00         Context:ensur for $196.00 dollers n
Matched: $204.00         Context: pump for $204.00 dollers w
```

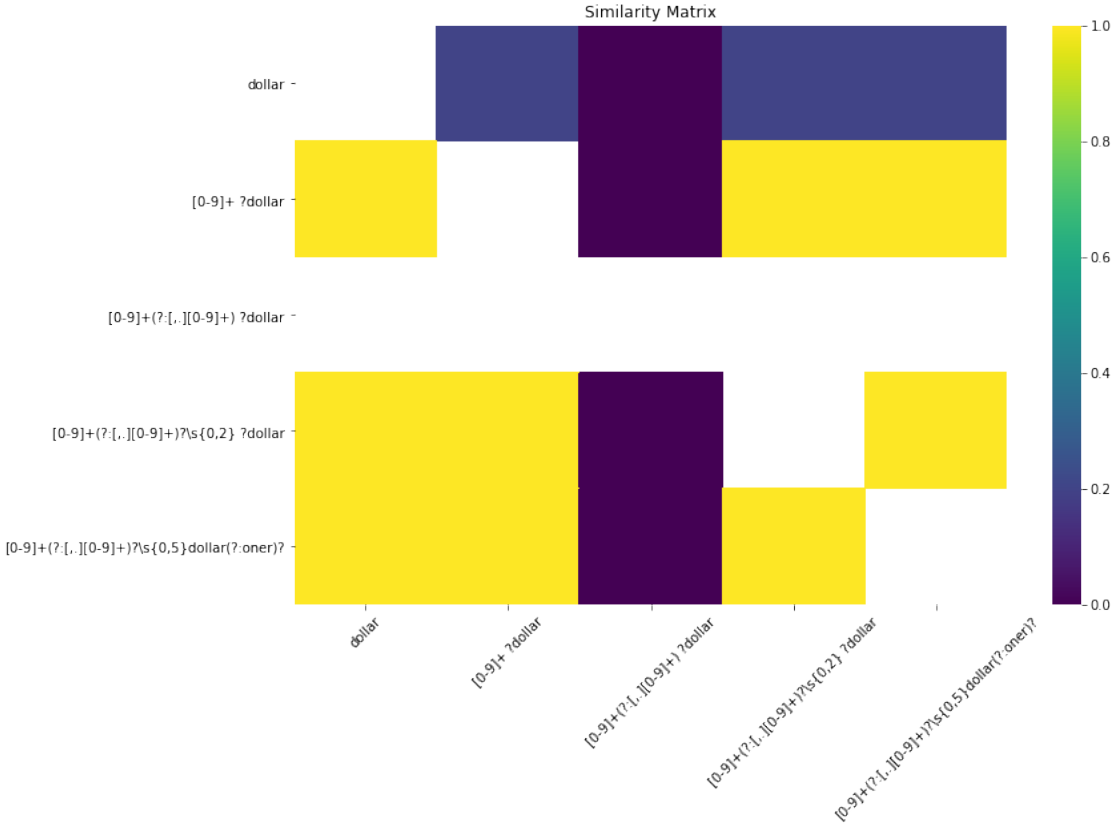
```
Matched: $9.00        Context:e charged $9.00 for the r
Matched: $100        Context:I ordered $100 plus in c
Matched: $400        Context:ying over $400 for two $
Matched: $45        Context:0 for two $45 tickets!
Matched: $148        Context:kets:        2 x $148 = $296.00
Matched: $296.00        Context: x $148 = $296.00
Service F
Matched: $48.84        Context: Fee:        2 x $48.84 = $97.68

Matched: $97.68        Context: $48.84 = $97.68
Delivery:
Matched: $15.00        Context:livery:                $15.00
Order Tot
Matched: $408.68        Context:er Total:        $408.68
I was ske
Matched: $79        Context:sed to be $79 USD and e
Matched: $50        Context:heck out. $50 rebate em
Matched: $8        Context:pping was $8!).  Sure
Matched: $15        Context:, but for $15 I expect
Matched: $99        Context:wn to the $99 service s
Matched: $10        Context:ases over $10. I recomm
Matched: $50        Context:dditional $50 on the or
Matched: $800        Context:ve to put $800 more down
Matched: $10        Context:es wanted $10. The only
Matched: $4721.20        Context: to buy a $4721.20 versus my
Matched: $1253.16        Context:ticket of $1253.16. Hey, tha
Matched: $3468.04        Context: an extra $3468.04!!!! Besid
['$20', '$49', '$2000', '$5', '10dollar', '$300', '$8,550', '$196.00', '$204.00', '$9.00', '$1
```

Ex.9.2.3 Use the .report() method. e_re.report(), and print the all patterns in the development process using the .pattern method - i.e. e_re.patterns

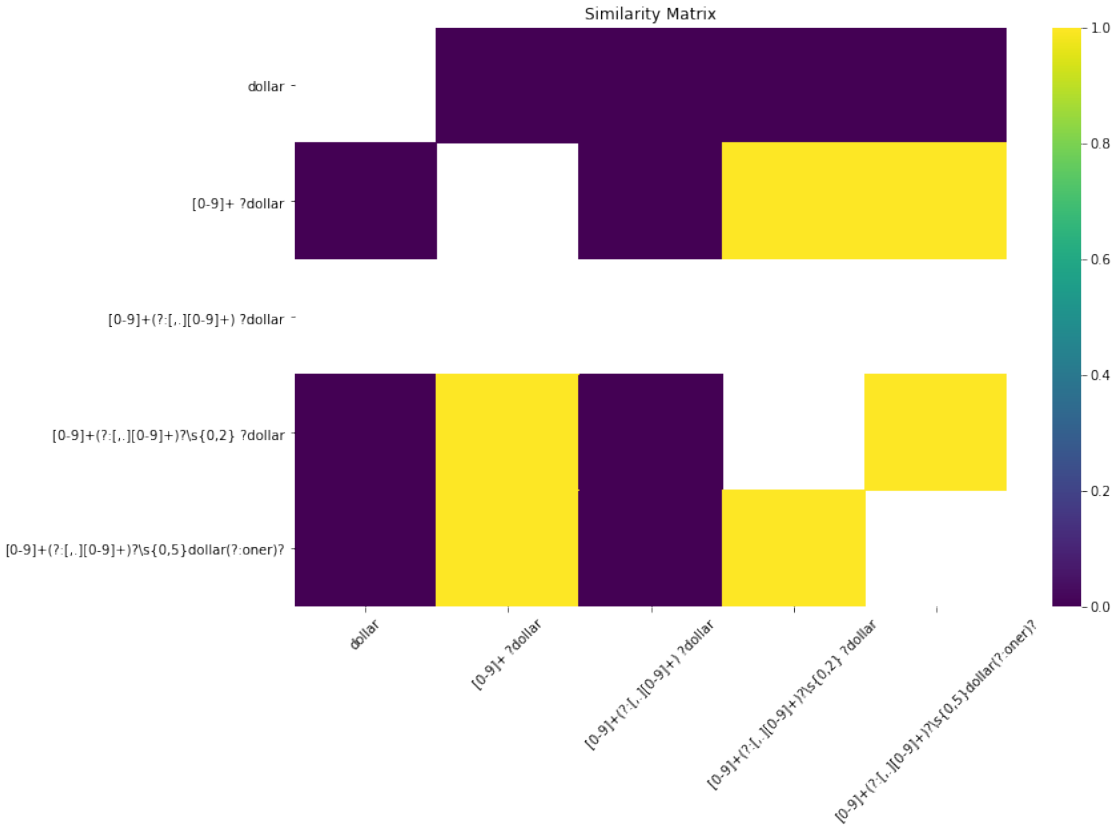
```
In [133]: # [Answer to Ex. 9.2.3] (only 'dollar', does not include '$')

          explore_money = e_re.ExploreRegex(sample_string)
          first = 'dollar'
          second = '[0-9]+ ?dollar'
          third = '[0-9]+(?:[,.][0-9]+) ?dollar'
          fourth = '[0-9]+(?:[,.][0-9]+)?\s{0,2} ?dollar'
          final = '[0-9]+(?:[,.][0-9]+)?\s{0,5}dollar(?:oner)?'
          patterns = [first,second,third,fourth,final]

          for pattern in patterns:
              explore_money.explore_difference(pattern,patterns[0])
          explore_money.explore_pattern(final)

          explore_money.report('soft')
```

```
------ Pattern: dollar         Matched 10 patterns -----
Found 0 overlaps between the expressions:

        pattern1: dollar          and

        pattern2: dollar

        10 included in pattern1 and not in the pattern2

        10 was included in pattern2 and not in pattern1
------ Pattern: [0-9]+ ?dollar         Matched 2 patterns -----
Found 4 overlaps between the expressions:

        pattern1: [0-9]+ ?dollar          and

        pattern2: dollar

        0 included in pattern1 and not in the pattern2

        8 was included in pattern2 and not in pattern1
------ Pattern: [0-9]+(?:[,.][0-9]+) ?dollar         Matched 0 patterns -----
Found 0 overlaps between the expressions:

        pattern1: [0-9]+(?:[,.][0-9]+) ?dollar          and

        pattern2: dollar

        0 included in pattern1 and not in the pattern2

        10 was included in pattern2 and not in pattern1
------ Pattern: [0-9]+(?:[,.][0-9]+)?\s{0,2} ?dollar         Matched 2 patterns -----
Found 4 overlaps between the expressions:

        pattern1: [0-9]+(?:[,.][0-9]+)?\s{0,2} ?dollar          and

        pattern2: dollar

        0 included in pattern1 and not in the pattern2

        8 was included in pattern2 and not in pattern1
------ Pattern: [0-9]+(?:[,.][0-9]+)?\s{0,5}dollar(?:oner)?         Matched 2 patterns -----
Found 4 overlaps between the expressions:

        pattern1: [0-9]+(?:[,.][0-9]+)?\s{0,5}dollar(?:oner)?          and

        pattern2: dollar

        0 included in pattern1 and not in the pattern2
```

```
        8 was included in pattern2 and not in pattern1
Match: 3000 dollar        Context:rdering a 3000 dollar watch or
Match: 10dollar        Context: belts at 10dollarmall for p
------ Pattern: dollar         Matched 10 patterns -----
------ Pattern: [0-9]+ ?dollar         Matched 2 patterns -----
------ Pattern: [0-9]+(?:[,.][0-9]+) ?dollar         Matched 0 patterns -----
------ Pattern: [0-9]+(?:[,.][0-9]+)?\s{0,2} ?dollar         Matched 2 patterns -----
------ Pattern: [0-9]+(?:[,.][0-9]+)?\s{0,5}dollar(?:oner)?         Matched 2 patterns -----
```




```
In [134]: explore_money.report('hard')

------ Pattern: dollar         Matched 10 patterns -----
------ Pattern: [0-9]+ ?dollar         Matched 2 patterns -----
------ Pattern: [0-9]+(?:[,.][0-9]+) ?dollar         Matched 0 patterns -----
------ Pattern: [0-9]+(?:[,.][0-9]+)?\s{0,2} ?dollar         Matched 2 patterns -----
------ Pattern: [0-9]+(?:[,.][0-9]+)?\s{0,5}dollar(?:oner)?         Matched 2 patterns -----
```


Similarity Matrix
dollar
[0-9]+ ?dollar
[0-9]+(?:[,.][0-9]+) ?dollar
[0-9]+(?:[,.][0-9]+)?\s{0,2} ?dollar
[0-9]+(?:[,.][0-9]+)?\s{0,5}dollar(?:oner)?
dollar
[0-9]+ ?dollar
[0-9]+(?:[,.][0-9]+) ?dollar
[0-9]+(?:[,.][0-9]+)?\s{0,2} ?dollar
[0-9]+(?:[,.][0-9]+)?\s{0,5}dollar(?:oner)?
1.0
0.8
0.6
0.4
0.2
0.0