Lab. 14 - Raporty i wizualizacja danych

January 18, 2022

1 Lab. 14 - Raporty i wizualizacja danych

1.1 Kontrolki interaktywne w Jupyter Notebook

czyli jak korzystać z interaktywnych widgetów IPython, aby usprawnić eksplorację i analizę danych.

W eksploracji danych mało efektywne i zakłócające płynność analizy danych jest wielokrotne uruchamianie tej samej komórki, za każdym razem nieznacznie zmieniając parametry wejściowe, np. wybierając inną wartość funkcji, inne zakresy dat do analizy, czy motyw wizualizacji plotly.

Jednym z rozwiązań tego problemu są interaktywne kontrolki umożliwiające zmianę danych wejściowych bez konieczności przepisywania lub ponownego uruchamiania kodu. Mowa o **IPython widgets** (z biblioteki ipywidgets), które można zbudować za pomocą jednej linii kodu. Biblioteka pozwala nam przekształcić statyczne dokumenty Jupyter Notebook w interaktywne pulpity, idealne do eksploracji i wizualizacji danych.

1. Instalacja:

```
pip install ipywidgets
pip install pyarrow
pip install fastparquet
pip install chart_studio
```

1. Aktywacja widgetów dla Jupyter Notebook:

```
jupyter nbextension enable --py widgetsnbextension
```

2. Import:

```
import ipywidgets as widgets
from ipywidgets import interact, interact_manual
```

[1]:
```
import ipywidgets as widgets
from ipywidgets import interact, interact_manual
import chart_studio.plotly as py
```

[2]:
```
import pandas as pd
import numpy as np
```

Pokaż wszystkie dane wyjściowe komórek

[3]:
```
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = 'all'
```

1.2 Podstawowe widgety

Przed rozpoczęciem wykonywania głównego zadania, proszę zapoznać się z dokumentacją dostępną tutaj. Zawiera ona przykłady najprostszych widgetów. Dostępna jest również wersja interaktywna, proszę ją wykonać.

Tutaj znajduje się lista widgetów. Niech to będzie punkt odniesienia w trakcie pracy z wykorzystaniem widgetów.
Cała dokumentacja jest dostępna tutaj.

1.3 Wczytanie danych

Dane pochodzą z repozytorium Willa Koehrsena i zawierają statystyki dotyczące jego artykułów.

Więcej informacji znajdziesz tutaj.

[4]:
```
data = pd.read_parquet('https://github.com/WillKoehrsen/Data-Analysis/blob/
↪master/medium/data/medium_data_2019_01_26?raw=true')
data.head()
```

[4]:
```
     claps  days_since_publication  fans  \
129      2              597.301123     2
125     18              589.983168     3
132     51              577.363292    20
126      0              576.520688     0
121      0              572.533035     0

                                                  link  num_responses  \
129  https://medium.com/p/screw-the-environment-but…              0
125  https://medium.com/p/the-vanquishing-of-war-pl…              0
132  https://medium.com/p/capstone-project-mercedes…              0
126  https://medium.com/p/home-of-the-scared-5af0fe…              0
121  https://medium.com/p/the-triumph-of-peace-f485…              0

    publication      published_date  read_ratio  read_time  reads  …  \
129        None 2017-06-10 14:25:00       42.17          7     70  …
125        None 2017-06-17 22:02:00       30.34         14     54  …
132        None 2017-06-30 12:55:00       20.02         42    222  …
126        None 2017-07-01 09:08:00       35.85          9     19  …
121        None 2017-07-05 08:51:00        8.47         14      5  …

          type views word_count claps_per_word  editing_days <tag>Education  \
129  published   166       1859       0.001076             0              0
125  published   178       3891       0.004626             0              0
132  published  1109      12025       0.004241             0              0
126  published    53       2533       0.000000             0              0
121  published    59       3892       0.000000             1              0

     <tag>Data Science  <tag>Towards Data Science  <tag>Machine Learning  \
```

```
129                  0                                0                    0
125                  0                                0                    0
132                  0                                0                    1
126                  0                                0                    0
121                  0                                0                    0

     <tag>Python
129            0
125            0
132            1
126            0
121            0

[5 rows x 25 columns]
```

Proszę zapoznać się z danymi (.columns, .describe(), .info()).

[5]:
```
data.columns
```

[5]:
```
Index(['claps', 'days_since_publication', 'fans', 'link', 'num_responses',
       'publication', 'published_date', 'read_ratio', 'read_time', 'reads',
       'started_date', 'tags', 'text', 'title', 'title_word_count', 'type',
       'views', 'word_count', 'claps_per_word', 'editing_days',
       '<tag>Education', '<tag>Data Science', '<tag>Towards Data Science',
       '<tag>Machine Learning', '<tag>Python'],
      dtype='object')
```

[6]:
```
data.describe()
```

[6]:
```
              claps  days_since_publication         fans  num_responses  \
count    133.000000              133.000000   133.000000     133.000000
mean    1815.263158              248.407273   352.052632       7.045113
std     2449.074661              179.370879   479.060117       9.056108
min        0.000000                1.218629     0.000000       0.000000
25%      121.000000               74.543822    23.000000       0.000000
50%      815.000000              245.416130   136.000000       4.000000
75%     2700.000000              376.080598   528.000000      12.000000
max    13600.000000              597.301123  2588.000000      59.000000

       read_ratio   read_time         reads  title_word_count          views  \
count  133.000000  133.000000    133.000000        133.000000     133.000000
mean    29.074662   12.917293   6336.300752          7.127820   23404.030075
std     12.417670    9.510795   9007.284726          3.158475   33995.636496
min      8.110000    1.000000      1.000000          2.000000       3.000000
25%     20.020000    8.000000    363.000000          5.000000    1375.000000
50%     27.060000   10.000000   2049.000000          7.000000    7608.000000
75%     34.910000   14.000000   7815.000000          8.000000   30141.000000
max     74.370000   54.000000  41978.000000         16.000000  173714.000000
```

```
          word_count  claps_per_word  editing_days  <tag>Education  \
count     133.000000      133.000000    133.000000      133.000000
mean     3029.120301        0.957638     20.330827        0.729323
std      2393.414456        1.846756     74.111579        0.445989
min       163.000000        0.000000    -13.000000        0.000000
25%      1653.000000        0.052115      0.000000        0.000000
50%      2456.000000        0.421525      1.000000        1.000000
75%      3553.000000        1.099366      5.000000        1.000000
max     15063.000000       17.891817    349.000000        1.000000

       <tag>Data Science  <tag>Towards Data Science  <tag>Machine Learning  \
count         133.000000                  133.000000              133.000000
mean            0.609023                    0.436090                0.383459
std             0.489814                    0.497774                0.488067
min             0.000000                    0.000000                0.000000
25%             0.000000                    0.000000                0.000000
50%             1.000000                    0.000000                0.000000
75%             1.000000                    1.000000                1.000000
max             1.000000                    1.000000                1.000000

       <tag>Python
count   133.000000
mean      0.315789
std       0.466587
min       0.000000
25%       0.000000
50%       0.000000
75%       1.000000
max       1.000000
```

[7]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 133 entries, 129 to 17
Data columns (total 25 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   claps                      133 non-null    int64
 1   days_since_publication     133 non-null    float64
 2   fans                       133 non-null    int64
 3   link                       133 non-null    object
 4   num_responses              133 non-null    int64
 5   publication                133 non-null    object
 6   published_date             133 non-null    datetime64[ns]
 7   read_ratio                 133 non-null    float64
 8   read_time                  133 non-null    int64
 9   reads                      133 non-null    int64
```

```
 10  started_date                133 non-null    datetime64[ns]
 11  tags                        133 non-null    object
 12  text                        133 non-null    object
 13  title                       133 non-null    object
 14  title_word_count            133 non-null    int64
 15  type                        133 non-null    object
 16  views                       133 non-null    int64
 17  word_count                  133 non-null    int64
 18  claps_per_word              133 non-null    float64
 19  editing_days                133 non-null    int32
 20  <tag>Education              133 non-null    int64
 21  <tag>Data Science           133 non-null    int64
 22  <tag>Towards Data Science   133 non-null    int64
 23  <tag>Machine Learning       133 non-null    int64
 24  <tag>Python                 133 non-null    int64
dtypes: datetime64[ns](2), float64(3), int32(1), int64(13), object(6)
memory usage: 26.5+ KB
```

Stwórz nowy DataFrame, z kolumnami: title, tags, published_date, publication, reads, views, word_count, claps, fans, read_time.

[8]:
```
df = data[['title', 'tags', 'published_date', 'publication', 'reads', 'views',␣
↪'word_count', 'claps', 'fans', 'read_time']]
```

1.4 Wyświetlanie danych

Za pomocą *.loc* wyświetl artykuły, które zostały przeczytane więcej niż 1000 razy.

[9]:
```
df.loc[df['claps'] > 1000, ['title', 'claps']]
```

[9]:

	title	claps
110	Random Forest in Python	4900
113	Random Forest Simple Explanation	3000
108	Hyperparameter Tuning the Random Forest in Python	2100
104	Time Series Analysis in Python: An Introduction	4100
101	Stock Analysis in Python	2800
100	Stock Prediction in Python	7300
95	Correlation vs. Causation: An Example	1200
94	Learn By Sharing	1600
90	Overfitting vs. Underfitting: A Conceptual Exp…	1600
91	Overfitting vs. Underfitting: A Complete Example	1100
98	How to Master New Skills	4300
93	Statistical Significance Explained	4970
88	Python is the Perfect Tool for any Problem	12900
89	Markov Chain Monte Carlo in Python	5100
83	Bayes' Rule Applied	6000
82	Beyond Accuracy: Precision and Recall	7000
84	Controlling the Web with Python	7000

```
79   Data Visualization with Bokeh in Python, Part …   2700
81   Data Visualization with Bokeh in Python, Part …   1100
80              Histograms and Density Plots in Python   3000
74   Data Visualization with Bokeh in Python, Part …   2600
76         Visualizing Data with Pairs Plots in Python   2900
75          Introduction to Bayesian Linear Regression   5500
72   Bayesian Linear Regression in Python: Using Ma…   1300
65   Web Scraping, Regular Expressions, and Data Vi…   8000
69   A Complete Machine Learning Project Walk-Throu…  13600
67   A Complete Machine Learning Walk-Through in Py…   4100
71   A Complete Machine Learning Walk-Through in Py…   2200
77   Automated Machine Learning on the Cloud in Python   3900
62   Machine Learning Kaggle Competition Part One: …   6400
66              Automated Feature Engineering in Python   8500
58   Machine Learning Kaggle Competition Part Two: …   1300
63   A Feature Selection Tool for Machine Learning …   5500
64   A Conceptual Explanation of Bayesian Hyperpara…   2400
70   An Introductory Example of Bayesian Optimizati…   2800
56   Automated Machine Learning Hyperparameter Tuni…   2400
60   Why Automated Feature Engineering Will Change …   3300
51   The most important part of a data science proj…   4400
54   A "Data Science for Good" Machine Learning Pro…   3800
57   An Implementation and Explanation of the Rando…   1100
49            Practical Advice for Data Science Writing   2900
50   Another Machine Learning Walk-Through and a Ch…   3100
41   Wikipedia Data Science: Working with the World…   3400
33                  Neural Network Embeddings Explained   2600
52   Simpson's Paradox: How to Prove Opposite Argum…   2100
38                    My Weaknesses as a Data Scientist   3300
42      Recurrent Neural Networks by Example in Python   2500
45   Deploying a Keras Deep Learning Model as a Web…   2300
47                    Deploying a Python Web App on AWS   1400
39   Estimating Probabilities with Bayesian Modelin…   2800
27                    Python and Slack: A Natural Match   2100
25                          Jupyter Notebook Extensions   3800
19      The Next Level of Data Visualization in Python   7600
2        A Non-Technical Reading List for Data Science   1800
5    The Poisson Distribution and Poisson Process E…   2000
```

Teraz za pomocą *.loc* wyświetl artykuły, które zostały przeczytane więcej niż 500 razy.

[10]:
```
df.loc[df['claps'] > 500, ['title', 'claps']]
```

[10]:
```
                                                 title  claps
127  Facial Recognition Using Google's Convolutiona…    704
110                             Random Forest in Python   4900
113                   Random Forest Simple Explanation   3000
123       Improving the Random Forest in Python Part 1    865
```

```
108  Hyperparameter Tuning the Random Forest in Python   2100
..                                                  …    …
15                              The Disappearing Poor    626
19      The Next Level of Data Visualization in Python   7600
2        A Non-Technical Reading List for Data Science   1800
5    The Poisson Distribution and Poisson Process E…   2000
17           Interactive Controls in Jupyter Notebooks    901

[84 rows x 2 columns]
```

To najprostszy przykład pokazujący, że interaktywna zmiana parametrów jest potrzebna, by usprawnić proces analizy danych.
Można to zrobić na przykład za pomocą specjalnej metody, z dekoratorem *@interact*:

[11]:
```
from IPython.display import display, HTML
```

[12]:
```
@interact
def show_articles_more_than(column='claps', x=5000):
    display(HTML(f'<h3>Showing articles with more than {x} {column}<h3>'))
    display(df.loc[df[column] > x])
```

```
interactive(children=(Text(value='claps', description='column'),␣
↪IntSlider(value=5000, description='x', max=15…
```

Dokumentacja Interact. Tłumaczy m. in., w jaki sposób parametry funkcji są mapowane do widgetów.

Zauważ, że dekorator *@interact* automatycznie wywnioskował, że potrzebujemy pola tekstowego dla kolumny i suwaka int dla x.

Gdy potrzebujemy wymusić pewne ograniczenia interakcji, możemy ustawić dodatkowe opcje tworzonej funkcji, takie jak *dropdown* czy granice dla wielkości numerycznych - format to (start, stop, krok):

[13]:
```
@interact
def show_titles_more_than(column=list(df.select_dtypes('number').columns),
                          x=(1000, 5000, 100)):
    display(HTML(f'<h3>Showing articles with more than {x} {column}<h3>'))
    display(df.loc[df[column] > x])
```

```
interactive(children=(Dropdown(description='column', options=('reads', 'views',␣
↪'word_count', 'claps', 'fans',…
```

1.5 Dataframe explorer

Stwórz funkcję z dekoratorem *@interact*, żeby szybko znajdować korelację między dwoma wybranymi kolumnami.

[14]:
```
@interact
def column_correlation(column1=list(df.columns), column2=list(df.columns)):
    df_columns = df[[column1, column2]]
```

```
    display(HTML(f'<h3>Showing correlation of column {column1} and column␣
↪{column2}<h3>'))
    display(df_columns.corr())
```

```
interactive(children=(Dropdown(description='column1', options=('title', 'tags',␣
↪'published_date', 'publication…
```

Stwórz funkcję z dekoratorem *@interact*, żeby wywołać funkcję *describe()* dla wybranej kolumny.

[15]:
```
@interact
def describe_function(column=list(df.columns)):
    df_column = df[[column]]
    display(HTML(f'<h3>Showing describe() function of column {column}'))
    display(df_column.describe())
```

```
interactive(children=(Dropdown(description='column', options=('title', 'tags',␣
↪'published_date', 'publication'…
```

1.6 Widgety dla wykresów

Widgety interaktywne są szczególnie przydatne przy wybieraniu danych do wykresu. W tym wypadku również możemy użyć tego samego dekoratora *@interact*, z funkcjami wizualizującymi nasze dane.

Uwaga. Obiekt DataFrame nie ma metody iplot, jeśli nie jest połączony z plotly. Potrzebujemy **cufflinks**, aby połączyć pandas z plotly i dodać metodę iplot.

Dodatkowo, aby uniknąć uwierzytelniania, potrzebujemy trybu offline.

[16]:
```
import cufflinks as cf
cf.go_offline(connected=True)
cf.set_config_file(colorscale='plotly', world_readable=True)
```

[17]:
```
from plotly.offline import iplot, init_notebook_mode
init_notebook_mode(connected=True)
```

[18]:
```
@interact
def scatter_plot(x=list(df.select_dtypes('number').columns),
                 y=list(df.select_dtypes('number').columns)[1:]):
    df.iplot(kind='scatter', x=x, y=y, mode='markers',
             xTitle=x.title(), yTitle=y.title(), title=f'{y.title()} vs {x.
↪title()}')
```

```
interactive(children=(Dropdown(description='x', options=('reads', 'views',␣
↪'word_count', 'claps', 'fans', 'rea…
```

Dodaj więcej opcji do funkcji scatter_plot: 1. Parametr theme, korzystając z tych dostępnych w cf.themes.THEMES.keys() 2. Parametr colorscale, korzystając z tych dostępnych w cf.colors._scales_names.keys())

[19]:
```
@interact
def scatter_plot(x=list(df.select_dtypes('number').columns),
                 y=list(df.select_dtypes('number').columns)[1:],
                 theme=list(cf.themes.THEMES.keys()),
                 colorscale=list(cf.colors._scales_names.keys())):
    df.iplot(kind='scatter', x=x, y=y, mode='markers',
             xTitle=x.title(), yTitle=y.title(), title=f'{y.title()} vs {x.
 ↪title()}', theme=theme, colorscale=colorscale)
```

```
interactive(children=(Dropdown(description='x', options=('reads', 'views',␣
 ↪'word_count', 'claps', 'fans', 'rea…
```

Do funkcji scatter_plot dodaj parametr categories, grupujący nasze dane. Przetestuj jego działanie.

[20]:
```
df['binned_read_time'] = pd.cut(df['read_time'], bins=range(0, 56, 5))
df['binned_read_time'] = df['binned_read_time'].astype(str)

df['binned_word_count'] = pd.cut(df['word_count'], bins=range(0, 100001, 1000))
df['binned_word_count'] = df['binned_word_count'].astype(str)

categories=['binned_read_time', 'binned_word_count', 'publication']
```

[21]:
```
@interact
def scatter_plot(x=list(df.select_dtypes('number').columns),
                 y=list(df.select_dtypes('number').columns)[1:],
                 theme=list(cf.themes.THEMES.keys()),
                 colorscale=list(cf.colors._scales_names.keys()),
                 categories=list(categories)):
    df.iplot(kind='scatter', x=x, y=y, mode='markers',
             xTitle=x.title(), yTitle=y.title(), title=f'{y.title()} vs {x.
 ↪title()}', theme=theme,
             colorscale=colorscale, categories=categories)
```

```
interactive(children=(Dropdown(description='x', options=('reads', 'views',␣
 ↪'word_count', 'claps', 'fans', 'rea…
```

Być może zauważyłeś, że aktualizacja wykresu przebiegała powoli. W takim przypadku możesz użyć dekoratora *@interact_manual*, który dostarcza przycisku do aktualizacji.

Sprawdź działanie widgetu z dekoratorem *@interact_manual*.

[22]:
```
from ipywidgets import interact_manual
```

[23]:
```
@interact_manual
def scatter_plot(x=list(df.select_dtypes('number').columns),
                 y=list(df.select_dtypes('number').columns)[1:],
                 theme=list(cf.themes.THEMES.keys()),
                 colorscale=list(cf.colors._scales_names.keys()),
                 categories=list(categories)):
    df.iplot(kind='scatter', x=x, y=y, mode='markers',
```

```
            xTitle=x.title(), yTitle=y.title(), title=f'{y.title()} vs {x.
↪title()}', theme=theme,
            colorscale=colorscale, categories=categories)
```

```
interactive(children=(Dropdown(description='x', options=('reads', 'views',␣
↪'word_count', 'claps', 'fans', 'rea…
```

1.7 Własne widgety

Aby skorzystać jeszcze więcej z biblioteki ipywidgets, możemy sami tworzyć widgety i używać ich w funkcji interakcji.

Stwórz własny widget. Napisz funkcję stats_for_article_published_between, która pobiera datę początkową i końcową, oraz wyświetla statystyki dla wszystkich artykułów opublikowanych między nimi.

[24]:
```
df.set_index('published_date', inplace=True)
```

[25]:
```
df.index
```

[25]:
```
DatetimeIndex(['2017-06-10 14:25:00', '2017-06-17 22:02:00',
               '2017-06-30 12:55:00', '2017-07-01 09:08:00',
               '2017-07-05 08:51:00', '2017-07-19 21:09:00',
               '2017-07-25 17:54:00', '2017-07-27 21:17:00',
               '2017-07-30 17:50:00', '2017-08-01 14:21:00',
               …
               '2019-01-02 08:15:00', '2019-01-02 09:29:00',
               '2019-01-05 21:04:00', '2019-01-08 22:09:00',
               '2019-01-10 15:14:00', '2019-01-12 13:33:00',
               '2019-01-16 16:44:00', '2019-01-20 16:28:00',
               '2019-01-25 08:30:00', '2019-01-27 16:23:00'],
              dtype='datetime64[ns]', name='published_date', length=133,
freq=None)
```

[26]:
```
def stats_for_article_published_between(start_date, end_date):
    start_date = pd.Timestamp(start_date)
    end_date = pd.Timestamp(end_date)
    new_df = df.loc[(df.index >= start_date) & (df.index <= end_date)]
    num_articles = len(new_df)
    total_words = new_df['word_count'].sum()
    total_read_time = new_df['read_time'].sum()
    print(f'You published {num_articles} articles between {start_date.date()}␣
↪and {end_date.date()}.')
    print(f'These articles totalled {total_words:,} words and {total_read_time/
↪60:.2f} hours to read.')
```

Za pomocą następującego kodu funkcja staje się interaktywna:

[27]:
```
_ = interact(stats_for_article_published_between,
             start_date=widgets.DatePicker(value=pd.to_datetime('2018-01-01')),
             end_date=widgets.DatePicker(value=pd.to_datetime('2019-01-01')))
```

```
interactive(children=(DatePicker(value=Timestamp('2018-01-01 00:00:00'),␣
↪description='start_date'), DatePicker…
```

Napisz funkcję plot_up_to, aby narysować wykres kumulatywnej sumy wartości wybranej kolumny, do wybranego dnia.

Użyj Dropdown i DatePicker w funkcji *interact*.

[28]:
```
def plot_up_to(column, date):
    date = pd.Timestamp(date)
    plot_df = df.loc[df.index <= date].copy()
    plot_df[column].cumsum().iplot(mode='markers+lines',
                                   xTitle='published date',
                                   yTitle=column,
                                   title=f'Cumulative {column.title()} Until␣
↪{date.date()}')

_ = interact(plot_up_to, column=widgets.Dropdown(options=list(df.
↪select_dtypes('number').columns)),
             date = widgets.DatePicker(value=pd.to_datetime('2019-01-01')))
```

```
interactive(children=(Dropdown(description='column', options=('reads', 'views',␣
↪'word_count', 'claps', 'fans',…
```

1.8 Przeglądanie zdjęć

Stwórz funkcję z dekoratorem *@interact*, żeby przeglądać zdjęcia znajdujące się w wybranym folderze. Folder z 3-5 zdjęciami również umieść na repozytorium.

[29]:
```
import os
from IPython.display import Image
```

[30]:
```
fdir = "./cities/"

@interact
def show_images(file=os.listdir(fdir)):
    display(Image(fdir+file))
```

```
interactive(children=(Dropdown(description='file', options=('cracow.jpg',␣
↪'helsinki.jpg', 'prague.jpg', 'warsa…
```

1.9 Przeglądanie plików

Stwórz funkcję z dekoratorem *@interact*, żeby przeglądać pliki znajdujące się w wybranych folderach. Skorzystaj z następujących (przykładowych) opcji komendy *ls*: **ls -a -t -r -l -h**. Więcej informacji znajduje się tutaj.

[31]:
```
import subprocess
```

[32]:
```
%ls "./"

root_dir = './'
dirs = [d for d in os.listdir(root_dir) if not '.' in d]

@interact
def show_dir(dir=dirs):
    x = subprocess.check_output(f"cd {root_dir}{dir} && dir /s/w/o/p ",␣
 ↪shell=True)
    print(x)
```

```
 Volume in drive E has no label.
 Volume Serial Number is AE9A-C96D

 Directory of E:\Adrian-
studia\Semestr_5\AiBD_classroom\laboratorium-13-Juninho25

18.01.2022  15:25    <DIR>          .
18.01.2022  15:25    <DIR>          ..
18.01.2022  14:21    <DIR>          .ipynb_checkpoints
18.01.2022  14:17    <DIR>          cities
18.01.2022  14:17    <DIR>          images
18.01.2022  15:25            74˙000 Lab. 14 - Raporty i wizualizacja
danych.ipynb
               1 File(s)         74˙000 bytes
               5 Dir(s)  81˙638˙313˙984 bytes free
```

```
interactive(children=(Dropdown(description='dir', options=('cities', 'images'),␣
 ↪value='cities'), Output()), _d…
```

1.10 Zależne widgety

Jeśli chcemy opcje jednego widgetu uzależnić od wartości innego widgetu, używamy metody *observe*.

Wykorzystaj metodę *observe*, żeby zmienić funkcję przeglądania zdjęć tak, by móc wybierać zarówno ścieżkę, jak i obraz do wyświetlenia. Drugi folder z 3-5 zdjęciami również umieść na repozytorium.

[33]:
```
directory = widgets.Dropdown(options=['images', 'cities'])
images = widgets.Dropdown(options=os.listdir(directory.value))

def update_images(*args):
    images.options = os.listdir(directory.value)

directory.observe(update_images, 'value')

def show_images(fdir, file):
    display(Image(f'{fdir}/{file}'))
```

```
_ = interact(show_images, fdir=directory, file=images)
```

```
interactive(children=(Dropdown(description='fdir', options=('images', 'cities'),␣
↪value='images'), Dropdown(des…
```

Możemy również przypisać zmienną do outputu funkcji *interact*, a następnie ponownie użyć widgetu. Może mieć to jednak niezamierzone skutki!

Teraz zmiana wartości w jednej lokalizacji zmienia ją w obu miejscach! Może to być drobna niedogodność, ale zaletą jest to, że możemy ponownie wykorzystać interaktywny element.

[34]:
```
dependent_widget = interact(show_images, fdir=directory, file=images)
```

```
interactive(children=(Dropdown(description='fdir', options=('images', 'cities'),␣
↪value='images'), Dropdown(des…
```

[35]:
```
dependent_widget.widget
```

```
interactive(children=(Dropdown(description='fdir', options=('images', 'cities'),␣
↪value='images'), Dropdown(des…
```

1.11 Wnioski

Dzięki tej serii laboratorium poznałem nowe narzędzie, jakim są interaktywne widgety. Muszę przyznać, że jest to rozwiązanie bardzo wygodne oraz ułatwiające życie studenta, pozwala oszczędzić dużo czasu poświęconego na wpisywanie mnóstwa danych do przetestowania. Myślę, że od teraz, będę używał takich widgetów. Uważam te laboratoria za bardzo przyjmne, instrukcja była przejrzysta, wszystko krok po kroku zostało odpowiednio opisane oraz zostały udostępnione linki do odpowiednich materiałów. Jedynym minusem było przeznaczenie tego notatnika na system Linux, natomiast na systemie Windows jest trochę inaczej. W zadaniu z przeglądaniem plików była podana komenda 'ls', której odpowiednikiem na Windowsie jest 'dir' i ma trochę inne działanie i składnie. Po chwili szukania jednak ten problem bez przeszkód udało się rozwiązać.