SPRAWOZDANIE

Zajęcia: Eksploracja i wizualizacja danych
Prowadzący: prof. dr hab. Vasyl Martsenyuk

Laboratorium: 2
Temat:
" Graficzna wizualizacja danych z użyciem bibliotek `matplotlib` i `seaborn`"
Wariant: 2

Link do repozytorium: https://github.com/jozek24/Eiwd

Józef Salik
Informatyka II stopień,
niestacjonarne,
3 semestr

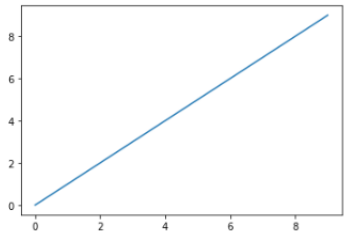
1. Polecenie:

Jedną z ważnych części analizy danych jest wizualizacja graficzna. Może to być częścią procesu badawczego - na przykład, aby pomóc zidentyfikować emisje lub wymagane przekształcanie danych lub jako sposób na generowanie pomysłów na modele. Python ma wiele dodatkowych bibliotek do tworzenia statycznych lub dynamicznych wizualizacji, ale skupimy się głównie na `matplotlib` i bibliotekach, które na nim budują.
Z biegiem czasu `matplotlib` stworzyło wiele dodatkowych zestawów narzędzi do wizualizacji danych, które wykorzystują `matplotlib` jako „rdzeń”. Jednym z takich narzędzi jest `seaborn`.

Zadanie dotyczy implementacji wszystkich możliwości tworzenia wykresów na podstawie tutorialu. Wariant jest określony zestawem danych.

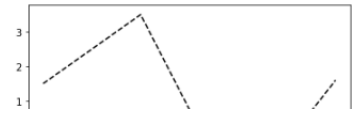
2. Wykonanie

```
In [1]: %matplotlib inline

        import matplotlib.pyplot as plt

In [4]: import numpy as np #import biblioteki numpy

In [5]: data = np.arange(10) #Stworzenie tablicy z 10 pkt
        data
Out[5]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

In [6]: plt.plot(data) #Stworzenie wykresu liniowego
Out[6]: [<matplotlib.lines.Line2D at 0x1c86ef97430>]
```

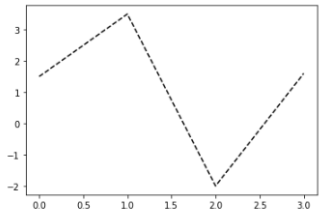


Rysunki i wykresy pomocnicze

```
In [10]: fig = plt.figure() #utworzenie obiektu Figure
         <Figure size 432x288 with 0 Axes>

In [11]: ax1 = fig.add_subplot(2, 2, 1) #dodanie wykresu pomocniczego za pomocq add_subplot / 2 x 2 (tj.zawiera maks 4 wykresy)
                                        #i wybieramy pirewszy z czterech wykresów

In [12]: ax2 = fig.add_subplot(2, 2, 2)

In [13]: ax3 = fig.add_subplot(2, 2, 3)

In [14]: ax4 = fig.add_subplot(2, 2, 4)

In [18]: plt.plot([1.5, 3.5, -2, 1.6], 'k--') #Polecenie kreślenia, na przykład plt.plot / k-- określa rodzaj linii
Out[18]: [<matplotlib.lines.Line2D at 0x1c86f2f2370>]
```



Out[18]: [<matplotlib.lines.Line2D at 0x1c86f2f2370>]



In [19]:
```
ax1.hist(np.random.randn(100), bins=20, color='k', alpha=0.3)
```

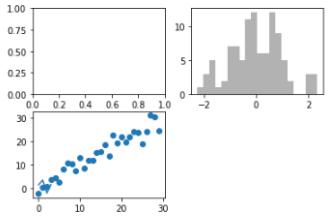
Out[19]:
```
(array([ 1.,  1.,  2.,  2.,  3.,  4.,  6.,  7.,  6., 10.,  6., 12., 11.,
         9.,  5.,  7.,  2.,  2.,  1.,  3.]),
 array([-2.93914425, -2.68362636, -2.42810848, -2.17259059, -1.9170727 ,
        -1.66155481, -1.40603693, -1.15051904, -0.89500115, -0.63948326,
        -0.38396538, -0.12844749,  0.1270704 ,  0.38258828,  0.63810617,
         0.89362406,  1.14914195,  1.40465983,  1.66017772,  1.91569561,
         2.1712135 ]),
 <BarContainer object of 20 artists>)
```

In [20]:
```
ax2.scatter(np.arange(30), np.arange(30) + 3 * np.random.randn(30))
```

Out[20]: <matplotlib.collections.PathCollection at 0x1c86f2ad940>

In [59]:
```
fig = plt.figure()
fig.add_subplot(2, 2, 1)
ax1 = fig.add_subplot(2, 2, 2)
ax2 = fig.add_subplot(2, 2, 3)
plt.plot([1.5, 3.5, -2, 1.6])
_ = ax1.hist(np.random.randn(100), bins=20, color='k', alpha=0.3)
ax2.scatter(np.arange(30), np.arange(30) + 3 * np.random.randn(30))
```

Out[59]: <matplotlib.collections.PathCollection at 0x1c872cb9550>

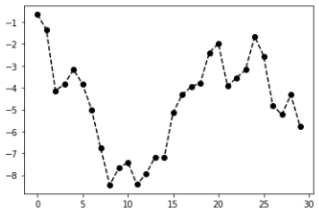


Kolor, znaczniki i style linii

In [22]:
```
from numpy.random import randn
```

In [31]:
```
plt.plot(randn(30).cumsum(), 'ko--')
```

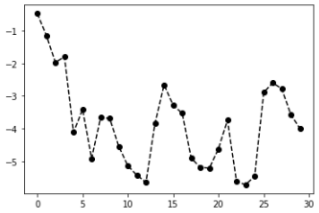
Out[31]:
```
[<matplotlib.lines.Line2D at 0x1c870756d60>]
```



to samo ale inaczej:

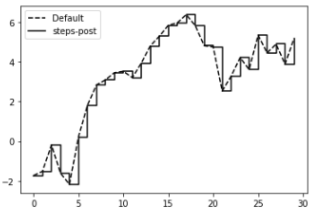
In [32]:
```
plt.plot(randn(30).cumsum(), color='k', linestyle='dashed', marker='o')
```

Out[32]:
```
[<matplotlib.lines.Line2D at 0x1c8707cc2b0>]
```



In [34]:
```
#zmiana interpolacji liniowej za pomocą parametru 'drawstyle'
data = np.random.randn(30).cumsum()
plt.plot(data, 'k--', label='Default')
plt.plot(data, 'k-', drawstyle='steps-post', label='steps-post')
plt.legend(loc='best')
```

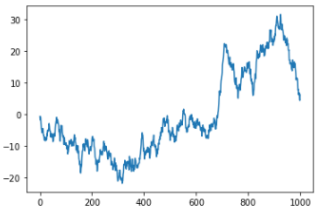
Out[34]:
```
<matplotlib.legend.Legend at 0x1c8708a7be0>
```



Etykiety osi, skala i legenda

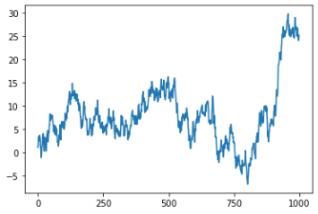
In [36]:
```
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax.plot(np.random.randn(1000).cumsum())
```

Out[36]:
```
[<matplotlib.lines.Line2D at 0x1c870968850>]
```



Zmiana etykiety na osi x: `set_xticks` a `set_xticklables`:

In [39]:
```
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax.plot(np.random.randn(1000).cumsum())
ticks = ax.set_xticks([0, 250, 500, 750, 1000])
#labels = ax.set_xticklabels(['one', 'two', 'three', 'four', 'five'], rotation=30, fontsize='small')
```



Parametr `rotation` obraca etykiety na osi x o 30 stopni. Ustaw `set_title` i `set_xlabel` dla osi x.

In [42]:
```
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax.plot(np.random.randn(1000).cumsum())
ticks = ax.set_xticks([0, 250, 500, 750, 1000])
#labels = ax.set_xticklabels(['one', 'two', 'three', 'four', 'five'], rotation=30, fontsize='small')
ax.set_title('Pierwszy wykres Jozef')
ax.set_xlabel('Kroki')
```

Out[42]:
```
Text(0.5, 0, 'Kroki')
```

Oś y jest modyfikowana dokładnie w ten sam sposób, tylko musisz zamienić `x` na `y` w powyższym kodzie. Klasa osi ma metodę `set`, która umożliwia grupowe ustawianie właściwości wykresu.

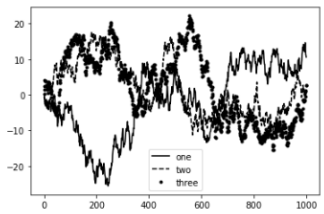
In [50]:
```
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax.plot(np.random.randn(1000).cumsum())
ticks = ax.set_yticks([-20, -10, 0, 10, 20, 30, 40])
props = {
            'title': 'Pierwszy wykres Jozef',
            'ylabel': 'oś y'
        }
ax.set(**props)
```

Out[50]:
```
[Text(0.5, 1.0, 'Pierwszy wykres Jozef'), Text(0, 0.5, 'oś y')]
```

Wyświetlenie legendy `label`

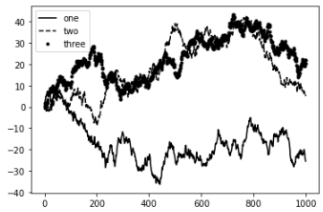
In [53]:
```
fig = plt.figure(); ax = fig.add_subplot(1, 1, 1)
ax.plot(randn(1000).cumsum(), 'k', label='one')
ax.plot(randn(1000).cumsum(), 'k--', label='two')
ax.plot(randn(1000).cumsum(), 'k.', label='three')
ax.legend(loc='best')
```

Out[53]:
```
<matplotlib.legend.Legend at 0x1c871d3bc10>
```



Zapisywanie rysunków do pliku

In [54]:
```
fig = plt.figure(); ax = fig.add_subplot(1, 1, 1)
ax.plot(randn(1000).cumsum(), 'k', label='one')
ax.plot(randn(1000).cumsum(), 'k--', label='two')
ax.plot(randn(1000).cumsum(), 'k.', label='three')
ax.legend(loc='best')
plt.savefig('figpath.png', dpi=400, bbox_inches='tight')
```



Funkcja `savefig` zapisuje nie tylko na dysku. Może zapisać wykres do dowolnego obiektu plikopodobnego, na przykład `BytesIO`:

In [56]:
```
from io import BytesIO
buffer = BytesIO()
plt.savefig(buffer)
plot_data = buffer.getvalue()
```

```
<Figure size 432x288 with 0 Axes>
```

Konstruowanie wykresów z pomocą `pandas` a `seaborn`

In [57]:
```
import numpy as np
import  pandas as pd
```

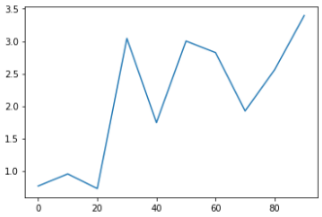
Wykresy liniowe

In [63]:
```
s = pd.Series(data=np.random.randn(10).cumsum(), index=np.arange(0, 100, 10)) #Domyślnie `plot()` tworzy wykresy liniowe
s.plot() #indeks obiektu s jest przekazywany do plot biblioteki matplotlib dla osi x
```

Wykresy liniowe

In [63]:
```
s = pd.Series(data=np.random.randn(10).cumsum(), index=np.arange(0, 100, 10)) #Domyślnie `plot()` tworzy wykresy liniowe
s.plot() #indeks obiektu s jest przekazywany do plot biblioteki matplotlib dla osi x
```

Out[63]: <AxesSubplot:>

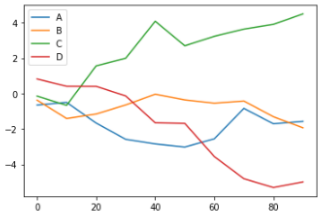


Metoda `plot` obiektu `DataFrame` wyświetla wykres dla każdej kolumny danych jako linię na tym samym wykresie podrzędnym, tworząc automatycznie legendę:

In [64]:
```
df = pd.DataFrame(np.random.randn(10, 4).cumsum(0), columns=['A', 'B', 'C', 'D'], index=np.arange(0, 100, 10))
```

In [65]:
```
df.plot()
```

Out[65]: <AxesSubplot:>



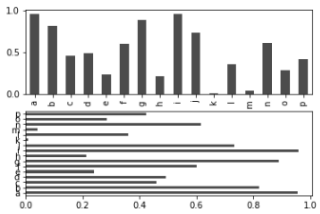
Wykresy kolumnowe

Metody `plot.bar()` i `plot.barh()` kreślą w pionie i poziome wykresy słupkowe. W tym przypadku indeksy obiektów `Series` i `DataFrame` jako etykiety na osi `x` (`bar`) lub `y` (`barh`).

Parametry `color = 'k'` i `alpha = 0.7` ustawiają kolor wykresu na czarny i częściową przezroczystość dla wypełnienia.

In [66]:
```
fig, axes = plt.subplots(2, 1)
data = pd.Series(np.random.rand(16), index=list('abcdefghijklmnop'))
data.plot.bar(ax=axes[0], color='k', alpha=0.7)
data.plot.barh(ax=axes[1], color='k', alpha=0.7)
```

Out[66]: <AxesSubplot:>



W `DataFrame` wykresy słupkowe grupują każdy wiersz wartości w grupę słupków odpowiadających każdej wartości w wierszu:

In [68]:
```
df = pd.DataFrame(np.random.rand(6, 4), index=['one', 'two', 'three', 'four', 'five', 'six'], columns=pd.Index(['A', 'B', 'C', 'D
```

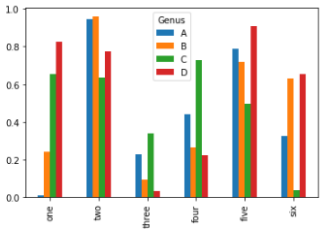
In [69]:
```
df
```

Out[69]:

Genus	A	B	C	D
one	0.011609	0.244304	0.654482	0.826471
two	0.944472	0.957914	0.635944	0.772819
three	0.228861	0.096569	0.339442	0.033297
four	0.443671	0.265465	0.728129	0.225287
five	0.786541	0.719092	0.495559	0.907727
six	0.327958	0.632740	0.040122	0.654916

In [70]:
```
df.plot.bar()
```

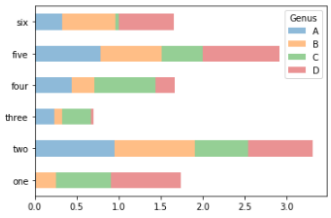
Out[70]: <AxesSubplot:>



Aby utworzyć skumulowane wykresy słupkowe, `DataFrame` jest ustawiane na `stacked = True`, co powoduje grupowanie wartości w każdym wierszu

In [71]:
```
df.plot.barh(stacked=True, alpha=0.5)
```

Out[71]: <AxesSubplot:>



Załóżmy, że mamy zestaw danych z rachunkami i napiwkami w restauracji i chcemy zbudować skumulowany wykres słupkowy pokazujący procent punktów danych dla każdej wielkości grupy każdego dnia. Załaduj dane z pliku tips.csv i stwórz podsumowanie według dnia i wielkości imprezy (Liczba osób):

In [74]:
```
df = pd.read_csv ( "IHME_GDP_1960_2050_Y2021M09D22.csv", encoding='latin1')
```

In [75]:
```
df.head()
```

Out[75]:

	location_id	location_name	iso3	level	year	gdp_ppp_mean	gdp_ppp_lower	gdp_ppp_upper	gdp_usd_mean	gdp_usd_lower	gdp_usd_upper
0	1	Global	G	Global	1960	1.748345e+13	1.601915e+13	1.911586e+13	1.296863e+13	1.266890e+13	1.334177e+13
1	1	Global	G	Global	1961	1.813537e+13	1.659537e+13	1.982493e+13	1.346097e+13	1.314767e+13	1.383021e+13
2	1	Global	G	Global	1962	1.895328e+13	1.739039e+13	2.061477e+13	1.406576e+13	1.376060e+13	1.443746e+13
3	1	Global	G	Global	1963	1.965662e+13	1.811706e+13	2.134993e+13	1.461831e+13	1.432132e+13	1.497693e+13

In [94]:
```
crosstab = pd.crosstab(dane['level'], dane['location_name'])
```

In [95]:
```
crosstab
```

Out[95]:

location_name	Afghanistan	Albania	Algeria	American Samoa	Andorra	Angola	Antigua and Barbuda	Argentina	Armenia	Australia	...	United States of America	Upper-middle income	Uruguay	Uzbekistan
level															
Country	91	91	91	91	91	91	91	91	91	91	...	91	0	91	91
GBD Super Region	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0
Global	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0
World Bank Income Group	0	0	0	0	0	0	0	0	0	0	...	0	91	0	0

4 rows × 216 columns

In [96]:
```
crosstab = crosstab.loc[:,:]
```

In [97]:
```
crosstab = crosstab.div(crosstab.sum(1), axis=0) #normalizacja danych
```

In [98]:
```
crosstab
```

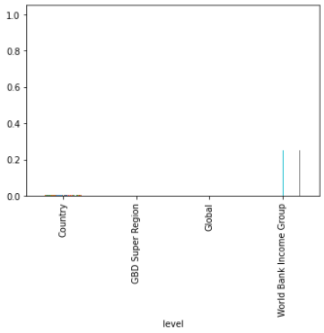
Out[98]:

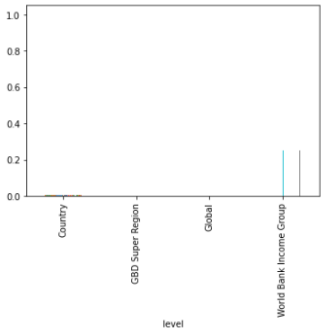
location_name	Afghanistan	Albania	Algeria	American Samoa	Andorra	Angola	Antigua and Barbuda	Argentina	Armenia	Australia	...	United States of America	Upper-middle income	Uruguay	Uzbeki
level															
Country	0.004902	0.004902	0.004902	0.004902	0.004902	0.004902	0.004902	0.004902	0.004902	0.004902	...	0.004902	0.00	0.004902	0.00
GBD Super Region	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.00	0.000000	0.00
Global	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.00	0.000000	0.00
World Bank Income Group	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.25	0.000000	0.00

4 rows × 216 columns

In [99]:
```
crosstab.plot.bar(legend=False)
crosstab.plot.bar(legend=False)
```

Out[99]: <AxesSubplot:xlabel='level'>





W przypadku, gdy musisz zagregować lub podsumować dane przed sporządzeniem wykresu, użycie pakietu `seaborn` może znacznie uprościć zadanie. Przyjrzyjmy się procentowi porad dziennie przy użyciu biblioteki `seaborn`:

In [100]:
```
import seaborn as sns
```

In [102]:
```
tips = pd.read_csv('tips.csv')
```

In [103]:
```
tips['tip_pct'] = tips['tip']/(tips['total_bill'] - tips['tip'])
```

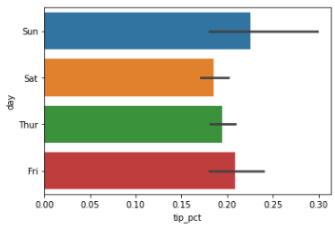
In [104]:
```
tips.head()
```

Out[104]:

	total_bill	tip	sex	smoker	day	time	size	tip_pct
0	16.99	1.01	Female	No	Sun	Dinner	2	0.063204
1	10.34	1.66	Male	No	Sun	Dinner	3	0.191244
2	21.01	3.50	Male	No	Sun	Dinner	3	0.199886
3	23.68	3.31	Male	No	Sun	Dinner	2	0.162494
4	24.59	3.61	Female	No	Sun	Dinner	4	0.172069

In [105]:
```
sns.barplot(x='tip_pct', y='day', data=tips, orient='h')
```

Out[105]: <AxesSubplot:xlabel='tip_pct', ylabel='day'>

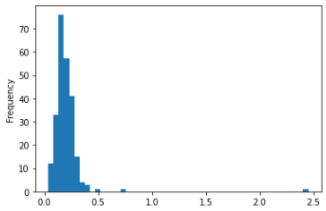


Histogramy i wykresy gęstości rozkładu

Histogram to rodzaj wykresu słupkowego, który dyskretnie wyświetla częstotliwość wartości.

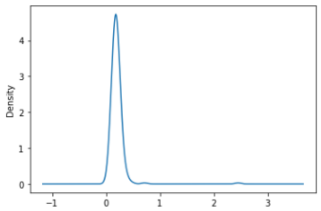
In [107]:
```
tips['tip_pct'].plot.hist(bins=50)
```

Out[107]:
```
<AxesSubplot:ylabel='Frequency'>
```



In [108]:
```
tips['tip_pct'].plot.kde()
```

Out[108]:
```
<AxesSubplot:ylabel='Density'>
```



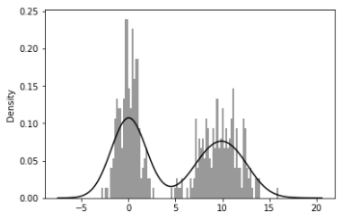
Biblioteka `seaborn` ułatwia tworzenie histogramów i wykresów gęstości za pomocą metody `distplot`, która pozwala na jednoczesne wykreślenie histogramu i ciągłej oceny gęstości.

In [109]:
```
comp1 = np.random.normal(0, 1, size=200)
comp2 = np.random.normal(10, 2, size=200)
values = pd.Series(np.concatenate([comp1, comp2]))
```

In [110]:
```
sns.distplot(values, bins=100, color='k')
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function an
d will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar fle
xibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[110]:
```
<AxesSubplot:ylabel='Density'>
```



Wykresy punktowe lub wykresy bitowe

Wykresy rozrzutu są przydatne do badania relacji między dwiema jednowymiarowymi seriami danych.

In [113]:
```
macro = pd.read_csv ( "IHME_GDP_1960_2050_Y2021M09D22.csv", encoding='latin1')
macro
```

Out[113]:

	location_id	location_name	iso3	level	year	gdp_ppp_mean	gdp_ppp_lower	gdp_ppp_upper	gdp_usd_mean	gdp_usd_lower	gdp_usd_upper
0	1	Global	G	Global	1960	1.748345e+13	1.601915e+13	1.911586e+13	1.296863e+13	1.266890e+13	1.334177e+13
1	1	Global	G	Global	1961	1.813537e+13	1.659537e+13	1.982493e+13	1.346097e+13	1.314767e+13	1.383021e+13
2	1	Global	G	Global	1962	1.895328e+13	1.739039e+13	2.061477e+13	1.406576e+13	1.376060e+13	1.443746e+13
3	1	Global	G	Global	1963	1.965662e+13	1.811706e+13	2.134993e+13	1.461831e+13	1.432132e+13	1.497693e+13
4	1	Global	G	Global	1964	2.100575e+13	1.935664e+13	2.276791e+13	1.552986e+13	1.523498e+13	1.587998e+13
...	...	...	...	...	...	...	...	...	...	...	...
19833	44578	Low income	NaN	World Bank Income Group	2046	3.617310e+12	3.140835e+12	4.166469e+12	1.149318e+12	1.031500e+12	1.271992e+12
19834	44578	Low income	NaN	World Bank Income Group	2047	3.724063e+12	3.225849e+12	4.292403e+12	1.186597e+12	1.061313e+12	1.318836e+12
19835	44578	Low income	NaN	World Bank Income Group	2048	3.831942e+12	3.307609e+12	4.424674e+12	1.224062e+12	1.092874e+12	1.365610e+12
19836	44578	Low income	NaN	World Bank Income Group	2049	3.941856e+12	3.398884e+12	4.560961e+12	1.262129e+12	1.122895e+12	1.413991e+12
19837	44578	Low income	NaN	World Bank Income Group	2050	4.053883e+12	3.482933e+12	4.713596e+12	1.300764e+12	1.151548e+12	1.457362e+12

19838 rows × 11 columns

In [115]:
```
macro=macro[macro['location_name'].isin(['Poland'])]
data = macro[['gdp_ppp_mean', 'gdp_ppp_lower','gdp_ppp_upper']]
data
```

Out[115]:

	gdp_ppp_mean	gdp_ppp_lower	gdp_ppp_upper
3913	6477.852541	3974.353115	8867.341510
3914	6854.160388	4209.334194	9266.617081
3915	6696.268564	4137.676353	9056.936779
3916	6981.908383	4381.353870	9390.144726
3917	7192.051449	4537.064677	9677.839801
...	...	...	...
3999	38341.602497	28096.663520	51449.445517
4000	38361.930630	27796.090370	52186.079877
4001	38342.127372	27466.932992	52654.694631
4002	38291.223003	27105.252343	53056.023696
4003	38239.561147	26728.543779	53156.930801

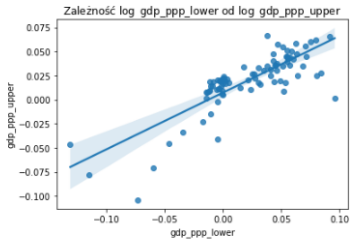
91 rows × 3 columns

In [116]:
```
trans_data = np.log(data).diff().dropna()
```

Teraz użyjmy funkcji `regplot` biblioteki `seaborn`, która kreśli wykresy punktowe i oferuje wykres regresji liniowej:

In [118]:
```
sns.regplot(x='gdp_ppp_lower', y='gdp_ppp_upper', data=trans_data)
plt.title('Zależność $\log$ {} od $\log$ {}'.format('gdp_ppp_lower', 'gdp_ppp_upper'))
```

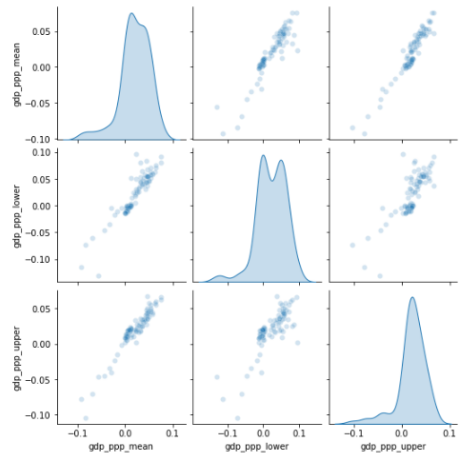
Out[118]:
```
Text(0.5, 1.0, 'Zależność $\\log$ gdp_ppp_lower od $\\log$ gdp_ppp_upper')
```



Podczas analizy danych przydatna jest możliwość przeglądania wszystkich wykresów rozrzutu w grupie zmiennych, tj. *wykresy sparowane* lub *macierz wykresów rozrzutu*. Biblioteka `seaborn` ma do tego wygodną funkcję `pairplot`, która w szczególności obsługuje umieszczanie histogramów lub oszacowań gęstości każdej zmiennej wzdłuż przekątnej:

In [119]:
```
sns.pairplot(trans_data, diag_kind='kde', plot_kws={'alpha': 0.2})
```

Out[119]:
```
<seaborn.axisgrid.PairGrid at 0x1c842d3a3d0>
```

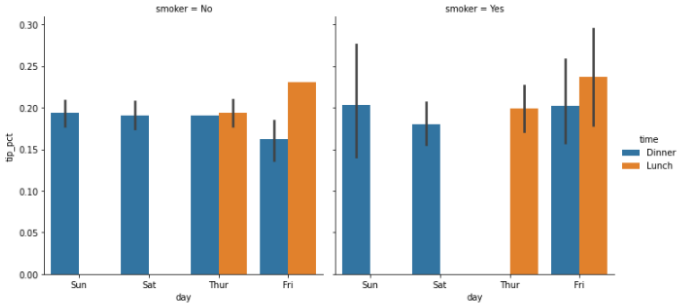


Dane kategoryczne

Jednym ze sposobów wizualizacji danych z wieloma zmiennymi kategorialnymi jest użycie *siatki aspektów*.

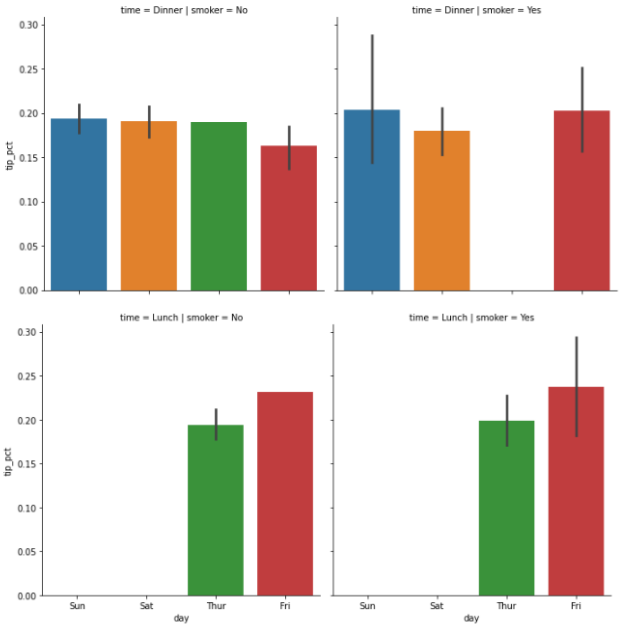
In [120]:
```
sns.catplot(x='day', y='tip_pct', hue='time', col='smoker', kind='bar', data=tips[tips.tip_pct < 1])
```

Out[120]:
```
<seaborn.axisgrid.FacetGrid at 0x1c842d90370>
```



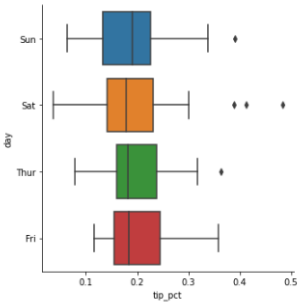
Zamiast wyświetlać różne kolory kolumn wykresu w aspekcie, możemy również rozszerzyć siatkę aspektów, dodając jeden wiersz naraz:

```
In [121]: sns.catplot(x='day', y='tip_pct', row='time', col='smoker', kind='bar', data=tips[tips.tip_pct < 1])
Out[121]: <seaborn.axisgrid.FacetGrid at 0x1c84347f730>
```



Funkcja `catplot` obsługuje inne rodzaje wykresów, które mogą być przydatne. Na przykład wykresy blokowe przedstawiające medianę, kwartyle i wartości odstające:

```
In [122]: sns.catplot(x='tip_pct', y='day', kind='box', data=tips[tips.tip_pct < 0.5])
Out[122]: <seaborn.axisgrid.FacetGrid at 0x1c843592610>
```



3. Wnioski:

- Matplotlib pozwala tworzyć wizualne wykresy
- Seaborn jest rozszerzeniem matplotlib’a
- Numpy pozwala generować tabele i macierze
- Wykresy matplotlib znajdują się w obiekcie Figure
- Pandas ma wbudowane metody ułatwiające renderowanie obiektów DataFrame i Series