ANALIZA SKŁADOWYCH GŁÓWNYCH

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sb
from sklearn.decomposition import PCA
```

Załadowanie zbioru danych:

```
data=pd.read_csv('HR_comma_sep.csv')
data
```

	satisfaction_level	last_evaluation	number_project	average_montly_hours	time_spend_company	Work_accident	left	promo
0	0.38	0.53	2	157	3	0	1	
1	0.80	0.86	5	262	6	0	1	
2	0.11	0.88	7	272	4	0	1	
3	0.72	0.87	5	223	5	0	1	
4	0.37	0.52	2	159	3	0	1	
...	...	...	...	...	...	...	...	
14994	0.40	0.57	2	151	3	0	1	
14995	0.37	0.48	2	160	3	0	1	
14996	0.37	0.53	2	143	3	0	1	
14997	0.11	0.96	6	280	4	0	1	
14998	0.37	0.52	2	158	3	0	1	

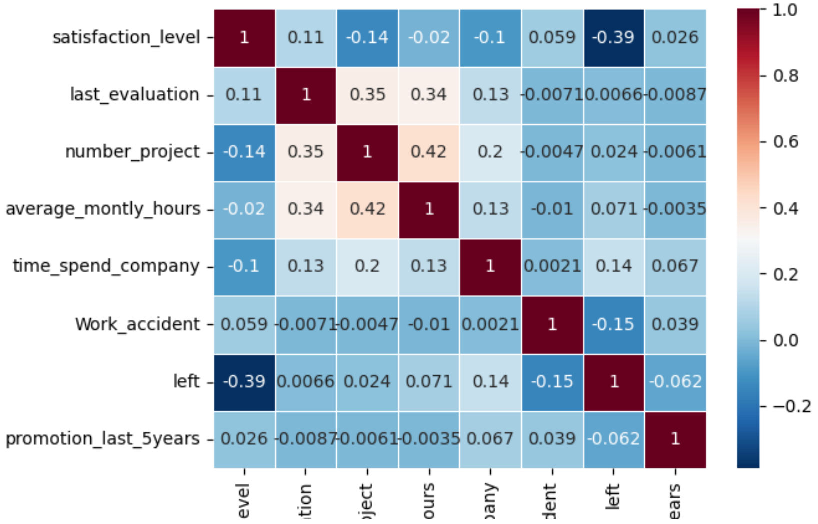
14999 rows × 10 columns

```
corr=data.corr()
```

```
/var/folders/wy/7w1mlgcx4vjfgbpvhlk2y7m00000gn/T/ipykernel_54143/2057684327.py:1: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future version, it will
  corr=data.corr()
```

```
sb.heatmap(corr,xticklabels=corr.columns,yticklabels=corr.columns,
            cmap='RdBu_r',
            annot=True,
            linewidth=0.5)
```

```
<AxesSubplot: >
```

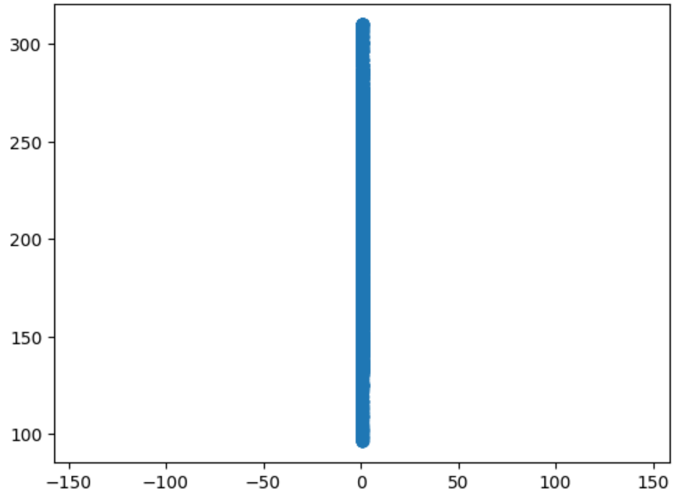



```
CorrMatrix = np.array(data.corr())
CorrMatrix
```

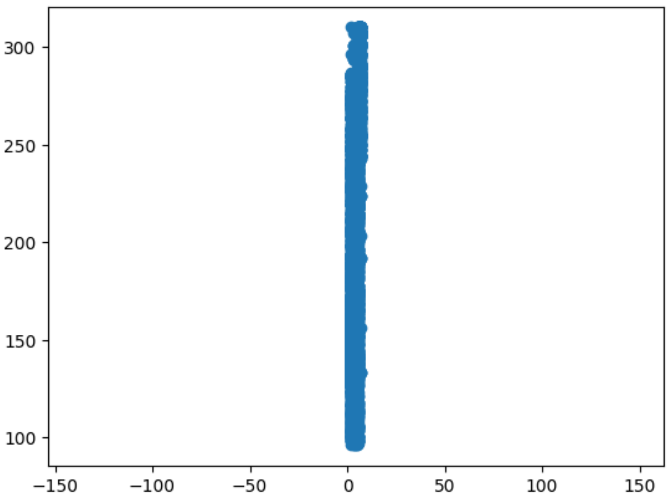
```
/var/folders/wy/7w1mlgcx4vjfgbpvhlk2y7m00000gn/T/ipykernel_54143/2744677357.py:1: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future version, it will
  CorrMatrix = np.array(data.corr())
array([[ 1.        ,  0.10502121, -0.14296959, -0.02004811, -0.10086607,
         0.05869724, -0.38837498,  0.02560519],
       [ 0.10502121,  1.        ,  0.34933259,  0.3397418 ,  0.13159072,
        -0.00710429,  0.00656712, -0.00868377],
       [-0.14296959,  0.34933259,  1.        ,  0.41721063,  0.19678589,
        -0.00474055,  0.02378719, -0.00606396],
       [-0.02004811,  0.3397418 ,  0.41721063,  1.        ,  0.12775491,
        -0.01014289,  0.07128718, -0.00354441],
       [-0.10086607,  0.13159072,  0.19678589,  0.12775491,  1.        ,
         0.00212042,  0.14482217,  0.06743293],
       [ 0.05869724, -0.00710429, -0.00474055, -0.01014289,  0.00212042,
         1.        , -0.15462163,  0.03924543],
       [-0.38837498,  0.00656712,  0.02378719,  0.07128718,  0.14482217,
        -0.15462163,  1.        , -0.06178811],
       [ 0.02560519, -0.00868377, -0.00606396, -0.00354441,  0.06743293,
         0.03924543, -0.06178811,  1.        ]])
```

```
plt.scatter(data.iloc[:,1], data.iloc[:,2])
plt.axis('equal');
plt.show()
```


```
plt.scatter(data.iloc[:,1], data.iloc[:,3])
plt.axis('equal');
plt.show()
```




```
plt.scatter(data.iloc[:,2], data.iloc[:,3])
plt.axis('equal');
plt.show()
```



```
w, v = np.linalg.eig(CorrMatrix)
print(w)
print(v)
```

```
[1.86091589 1.46409354 0.47745185 1.06058666 0.95598374 0.8454993
 0.62648811 0.7089809 ]
[[-0.18956186 -0.60825815  0.51043559  0.14578963 -0.2534991  -0.32268329
  -0.2910217   0.2433296 ]
 [ 0.46363715 -0.31222881 -0.27367838  0.15715943 -0.10307248 -0.06471173
   0.54777287  0.52257837]
 [ 0.55704703 -0.12254292  0.58883958  0.0129521   0.09858338  0.1887942
   0.24157676 -0.47335058]
 [ 0.52559587 -0.17853674 -0.30588994  0.11339814  0.0120681   0.25349244
  -0.72147388  0.02274205]
 [ 0.33395132  0.11709262 -0.11038416 -0.44415687 -0.04569912 -0.79303045
  -0.09314767 -0.16013636]
 [-0.06443923 -0.28140442  0.07016424 -0.42577604  0.81315664  0.06549289
  -0.02938544  0.25312908]
 [ 0.2163394   0.61631274  0.45356155  0.01069646  0.00816191  0.01364792
  -0.16219105  0.58392171]
 [-0.00870881 -0.11358933  0.03780465 -0.74989628 -0.50186771  0.39801173
   0.02283486  0.11154387]]
```

```
v[:,0]
```

```
array([-0.18956186,  0.46363715,  0.55704703,  0.52559587,  0.33395132,
       -0.06443923,  0.2163394 , -0.00870881])
```

```
v[:,1]
```

```
array([-0.60825815, -0.31222881, -0.12254292, -0.17853674,  0.11709262,
       -0.28140442,  0.61631274, -0.11358933])
```

```
v[:,2]
```

```
array([ 0.51043559, -0.27367838,  0.58883958, -0.30588994, -0.11038416,
        0.07016424,  0.45356155,  0.03780465])
```

```
v[:,3]
```

```
array([ 0.14578963,  0.15715943,  0.0129521 ,  0.11339814, -0.44415687,
       -0.42577604,  0.01069646, -0.74989628])
```

```
v[:,4]
```

```
array([-0.2534991 , -0.10307248,  0.09858338,  0.0120681 , -0.04569912,
        0.81315664,  0.00816191, -0.50186771])
```

```
v[:,5]
```

```
array([-0.32268329, -0.06471173,  0.1887942 ,  0.25349244, -0.79303045,
        0.06549289,  0.01364792,  0.39801173])
```

```
v[:,6]
```

```
array([-0.2910217 ,  0.54777287,  0.24157676, -0.72147388, -0.09314767,
       -0.02938544, -0.16219105,  0.02283486])
```

```
v[:,7]
```

```
array([ 0.2433296 ,  0.52257837, -0.47335058,  0.02274205, -0.16013636,
        0.25312908,  0.58392171,  0.11154387])
```

```
data2=data.drop("sales", axis='columns')
```

```
data2=data2.drop("salary", axis='columns')
```

Dwie składowe główne

Wyliczamy dwie składowe główne (**n_components=2**) czyli wektory bazowe nowego układu współrzędnych:

```
pca = PCA()
pca.fit(data2)
cumsum = np.cumsum(pca.explained_variance_ratio_)
d = np.argmax(cumsum >= 0.95) + 1
```

```
d
```

```
1
```

```
pca = PCA(n_components=2)
pca.fit(data2)
```

```
▾        PCA
PCA(n_components=2)
```

Wektory bazowe nowego układu wpółrzędnych:

```
print(pca.components_)
```

```
[[-1.00033243e-04  1.16454569e-03  1.03016936e-02  9.99939075e-01
   3.73905642e-03 -7.14335346e-05  6.08072674e-04 -1.02220131e-05]
 [-2.22798744e-02  1.53049805e-02  2.71231022e-01 -6.43334035e-03
   9.61232600e-01  2.85025356e-04  4.06830170e-02  6.19745208e-03]]
```

Wariancje dla nowych współrzędnych:

```
print(pca.explained_variance_)
```

```
[2.49461695e+03 2.17485649e+00]
```

Dodajemy do wykresu wektory wyznaczające nowy układ współrzędnych. Ich długość określona jest przez wariancje:

```
def draw_vector(v0, v1, ax=None):
  ax = ax or plt.gca()
  arrowprops=dict(arrowstyle='->',
```

```
    linewidth=2,
    shrinkA=0, shrinkB=0, color='black')
    ax.annotate('', v1, v0, arrowprops=arrowprops)
```

```
pca.explained_variance_
```

```
array([2.49461695e+03, 2.17485649e+00])
```

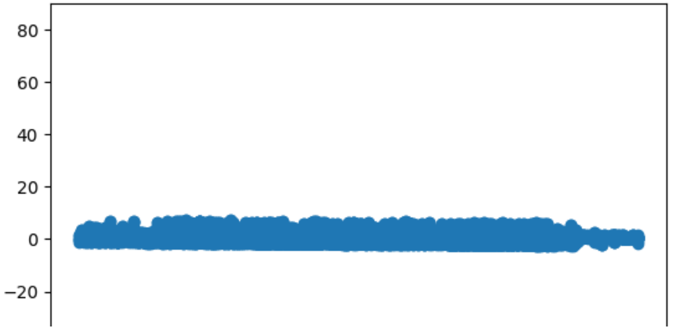
```
#plt.scatter(data2.iloc[:, 0], data2.iloc[:, 4], alpha=0.2)
#for length, vector in zip(pca.explained_variance_, pca.components_):
 # v = vector * 3 * np.sqrt(length)
 # draw_vector(pca.mean_, pca.mean_ + v)
#plt.axis('equal');
```

Współrzędne punktów w **nowym układzie współrzędnych**:

```
data_pca = pca.transform(data2)
data_pca
```

```
array([[-44.06780993,  -0.65140796],
       [ 60.96825747,   2.36617527],
       [ 70.98086581,   0.93751792],
       ...,
       [-58.06695598,  -0.5611184 ],
       [ 78.97016987,   0.61604457],
       [-43.0678815 ,  -0.65777155]])
```

```
plt.scatter(data_pca[:,0], data_pca[:,1])
plt.axis('equal');
```



Jedna składowa główna

Wyliczamy jedną składową główną (**n_components=1**) - chcemy wyeliminować jeden wymiar danych.

```
pca = PCA(n_components=1)
pca.fit(data2)
```

```
▾        PCA
PCA(n_components=1)
```

Współrzędne punktów w **nowym układzie współrzędnych**:

```
data_pca = pca.transform(data2)
```

Porówanie kształtów danych początkowych i po redukcji jednego wymiaru:

```
print("Początkowy shape: ", data2.shape)
print("Po transformacji shape:", data_pca.shape)

    Początkowy shape:  (14999, 8)
    Po transformacji shape: (14999, 1)
```

Dane początkowe i po redukcji wymiaru:

```
data_new = pca.inverse_transform(data_pca)
plt.scatter(data2.iloc[:, 1], data2.iloc[:, 2]),
plt.scatter(data_new[:, 0], data_new[:, 1]),
plt.axis('equal');
```