hubert_mentel_5

January 10, 2025

[1]:
```
import pandas as pd
import seaborn as sns
import plotly.express as px
import plotly.graph_objects as go
import matplotlib.pyplot as plt
import numpy as np
from sklearn.ensemble import IsolationForest
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

# Wczytanie pliku CSV
file_path = "gpu_data.csv"  # Upewnij się, że plik jest w tym samym katalogu co␣
 ↪skrypt
data = pd.read_csv(file_path)

# Zmiana ustawień wyświetlania, aby pokazać wszystkie wiersze
pd.set_option('display.max_rows', None)

# Wyświetlenie wszystkich wierszy
print(data)
```

	Model	Technology (nm)	Die Size (mm²)	ROP Units	TMU Units	\
0	GTX 750 Ti	28	148	16	40	
1	GTX 760	28	294	32	96	
2	GTX 770	28	294	32	128	
3	GTX 780	28	561	48	192	
4	GTX 780 Ti	28	561	48	240	
5	GTX 960	28	228	32	64	
6	GTX 970	28	398	56	104	
7	GTX 980	28	398	64	128	
8	GTX 980 Ti	28	601	96	176	
9	GTX 1050 Ti	14	132	32	48	
10	GTX 1060	16	200	48	80	
11	GTX 1070	16	314	64	120	
12	GTX 1070 Ti	16	314	64	152	
13	GTX 1080	16	314	64	160	
14	GTX 1080 Ti	16	471	88	224	
15	RTX 2060	12	445	48	120	

16	RTX 2060 Super	12	445	64	136
17	RTX 2070	12	445	64	144
18	RTX 2070 Super	12	545	64	160
19	RTX 2080	12	545	64	184
20	RTX 2080 Super	12	545	64	184
21	RTX 2080 Ti	12	754	88	272
22	RTX 3050	8	276	32	80
23	RTX 3060	8	276	48	112
24	RTX 3060 Ti	8	392	48	152
25	RTX 3070	8	392	64	184
26	RTX 3070 Ti	8	392	64	192
27	RTX 3080	8	628	96	272
28	RTX 3080 Ti	8	628	112	320
29	RTX 3090	8	628	112	328
30	RTX 4060	5	190	32	96
31	RTX 4060 Ti 8GB	5	190	32	128
32	RTX 4060 Ti 16GB	5	190	32	128
33	RTX 4070	5	295	48	184
34	RTX 4070 Ti	5	295	48	192
35	RTX 4080	5	380	64	304
36	RTX 4090	5	608	80	512

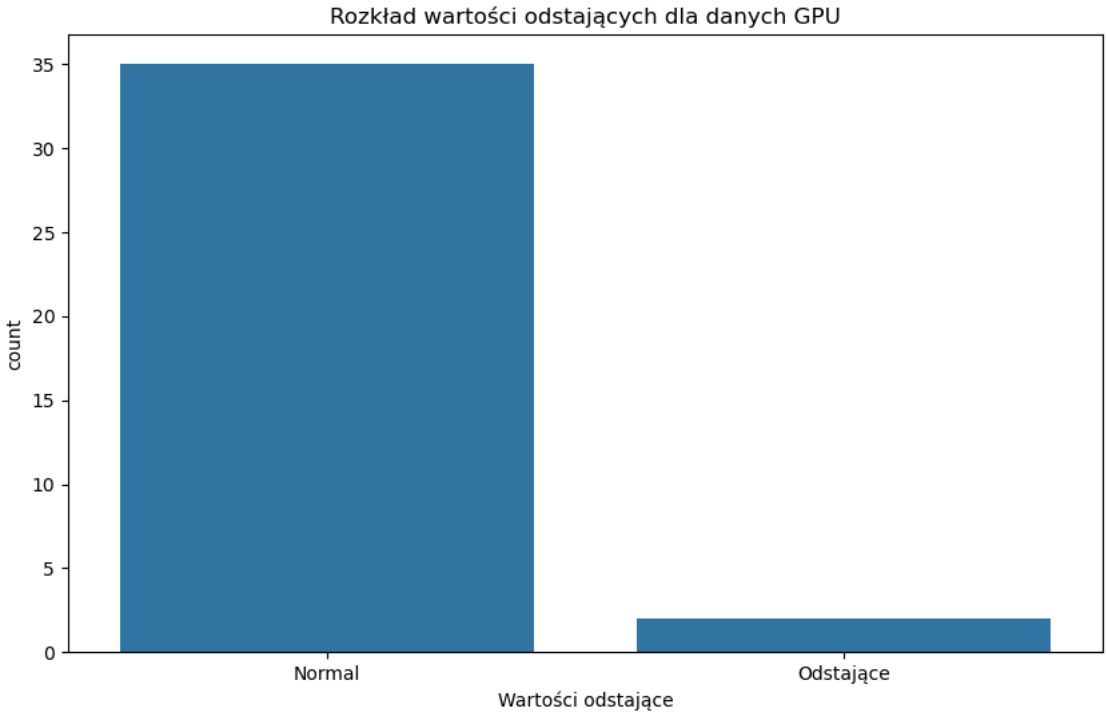
	CUDA Cores	Tensor Cores	Base Clock (MHz)	Boost Clock (MHz)	\
0	640	0	1020	1085	
1	1152	0	980	1033	
2	1536	0	1046	1085	
3	2304	0	863	900	
4	2880	0	875	928	
5	1024	0	1127	1178	
6	1664	0	1050	1178	
7	2048	0	1126	1216	
8	2816	0	1000	1075	
9	768	0	1290	1392	
10	1280	0	1506	1708	
11	1920	0	1506	1683	
12	2432	0	1607	1683	
13	2560	0	1607	1733	
14	3584	0	1480	1582	
15	1920	240	1365	1680	
16	2176	272	1470	1650	
17	2304	288	1410	1620	
18	2560	320	1605	1770	
19	2944	368	1515	1710	
20	3072	384	1650	1815	
21	4352	544	1350	1545	
22	2560	80	1552	1777	
23	3584	112	1320	1777	
24	4864	152	1410	1665	

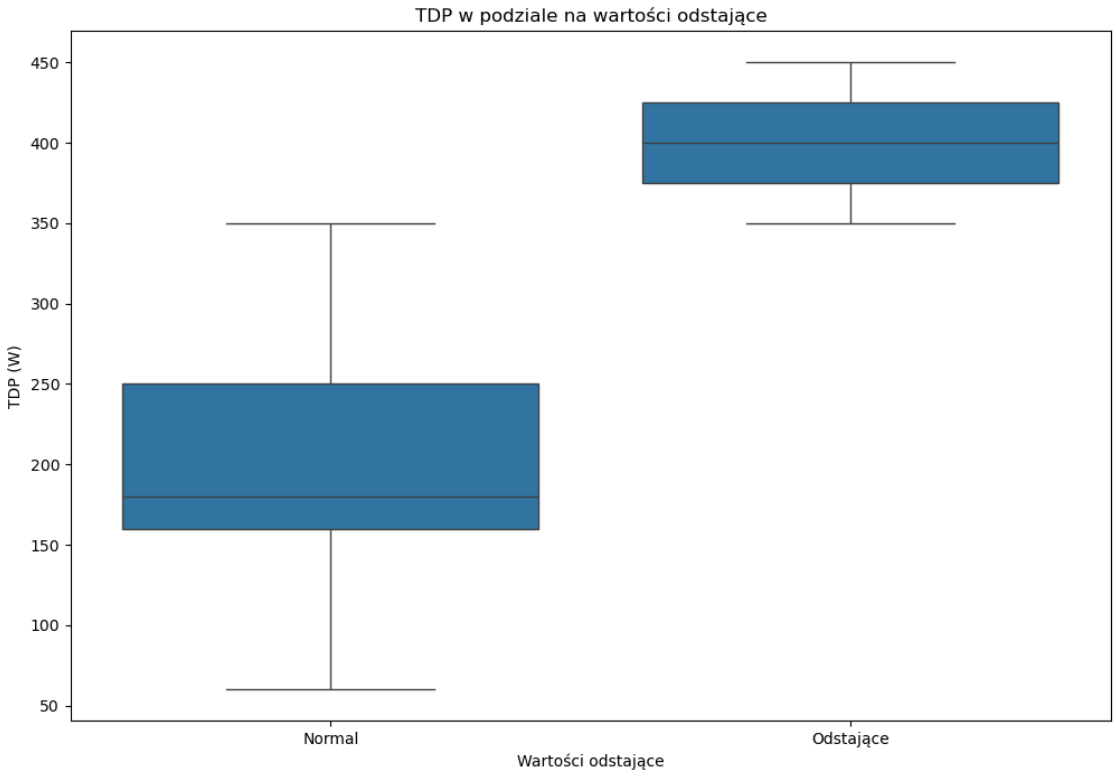
25	5888	184	1500	1725
26	6144	192	1575	1770
27	8704	272	1440	1710
28	10240	320	1365	1665
29	10496	328	1395	1695
30	3072	96	1830	2460
31	4352	128	2310	2535
32	4352	128	2310	2535
33	5888	184	1920	2475
34	7680	240	2310	2610
35	9728	304	2210	2510
36	16384	512	2235	2520

	Memory Size (GB)	Memory Speed (Gbps)	PCIe Version	TDP (W)	Price (USD)
0	2	5.4	3.0	60	149
1	2	6.0	3.0	170	249
2	2	7.0	3.0	230	399
3	3	6.0	3.0	250	499
4	3	7.0	3.0	250	699
5	2	7.0	3.0	120	199
6	4	7.0	3.0	145	329
7	4	7.0	3.0	165	549
8	6	7.0	3.0	250	649
9	4	7.0	3.0	75	139
10	6	8.0	3.0	120	249
11	8	8.0	3.0	150	379
12	8	8.0	3.0	180	449
13	8	10.0	3.0	180	599
14	11	11.0	3.0	250	699
15	6	14.0	3.0	160	349
16	8	14.0	3.0	175	399
17	8	14.0	3.0	175	499
18	8	14.0	3.0	215	499
19	8	14.0	3.0	215	699
20	8	15.5	3.0	250	699
21	11	14.0	3.0	250	999
22	8	14.0	4.0	130	249
23	12	15.0	4.0	170	329
24	8	14.0	4.0	200	399
25	8	14.0	4.0	220	499
26	8	19.0	4.0	290	599
27	10	19.0	4.0	320	699
28	12	19.0	4.0	350	1199
29	24	19.0	4.0	350	1499
30	8	17.0	4.0	115	299
31	8	18.0	4.0	160	399
32	16	18.0	4.0	160	499
33	12	21.0	4.0	200	599

34	12	21.0	4.0	285	799
35	16	22.4	4.0	320	1199
36	24	21.0	4.0	450	1599

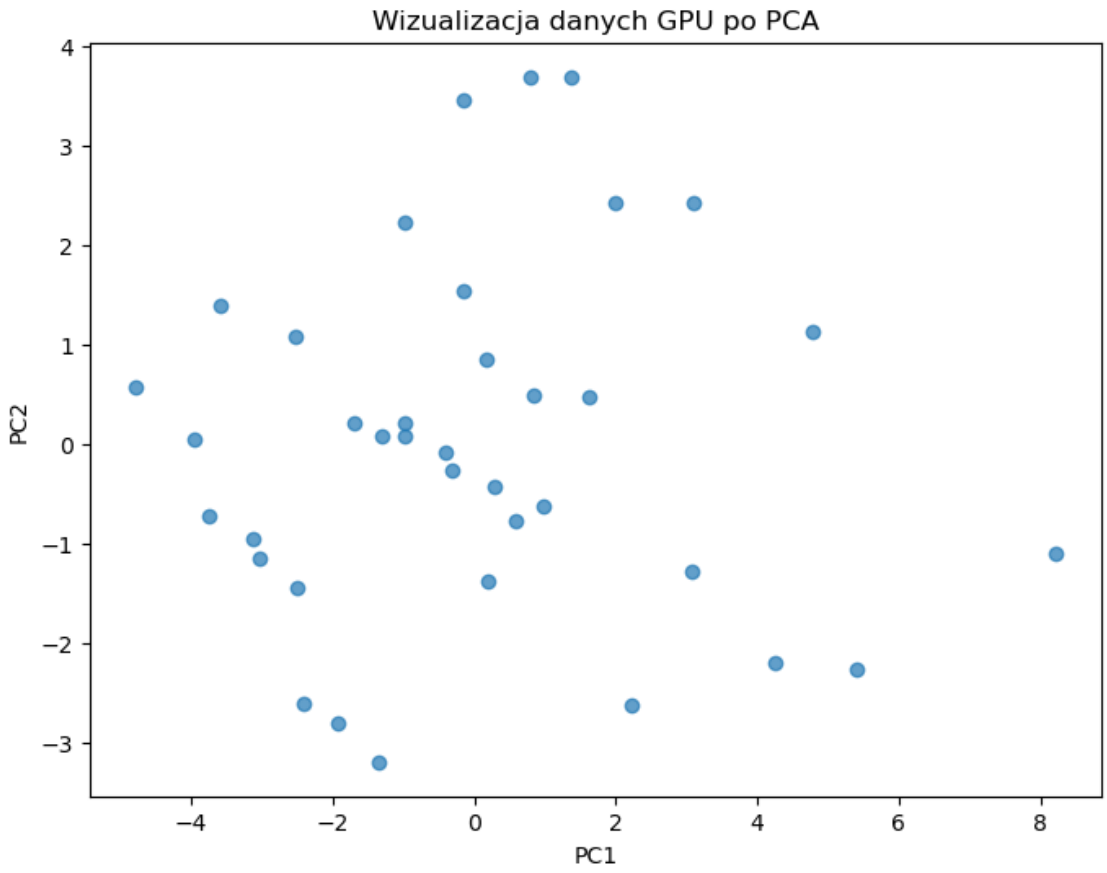
[3]:
```
# 1. Zidentyfikować wartości odstające za pomocą algorytmu Isolation Forest
df = pd.read_csv("gpu_data.csv")

# Wybierz tylko kolumny liczbowe do analizy
numerical_df = df.select_dtypes(include=['float64', 'int64'])

# Przeskalowanie danych
scaler = StandardScaler()
numerical_df_scaled = scaler.fit_transform(numerical_df)

# Model Isolation Forest
iso_forest = IsolationForest(contamination=0.05, random_state=42)
outliers = iso_forest.fit_predict(numerical_df_scaled)

# Oznaczenie wartości odstających (-1 oznacza odstające, 1 oznacza normalne)
df['Wartości odstające'] = np.where(outliers == -1, 'Odstające', 'Normal')

df.to_csv("gpu_data_with_outliers.csv", index=False)

# Wizualizacja rozkładu wartości odstających
plt.figure(figsize=(10, 6))
sns.countplot(data=df, x='Wartości odstające')
plt.title("Rozkład wartości odstających dla danych GPU")
plt.show()

# Wizualizacja zmiennych z uwzględnieniem wartości odstających
plt.figure(figsize=(12, 8))
sns.boxplot(data=df, x='Wartości odstające', y='TDP (W)')
plt.title("TDP w podziale na wartości odstające")
plt.show()
```


Rozkład wartości odstających dla danych GPU
count
0
5
10
15
20
25
30
35
Normal
Odstające
Wartości odstające

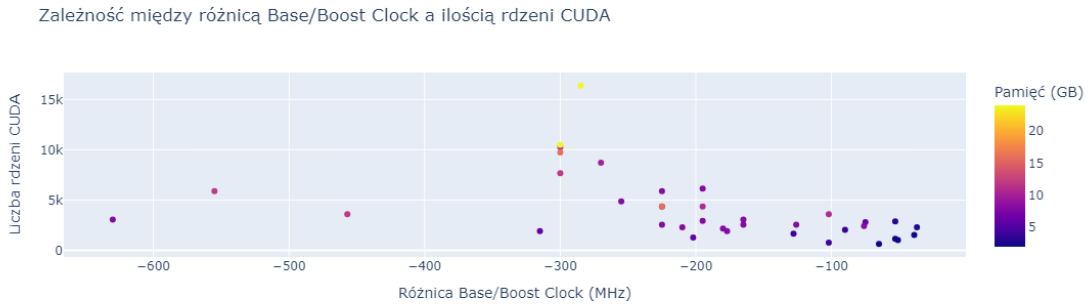

TDP w podziale na wartości odstające
TDP (W)
50
100
150
200
250
300
350
400
450
Normal
Odstające
Wartości odstające

[5]:
```
# 2. Redukować wymiarowość danych z użyciem PCA,

numerical_df = df.select_dtypes(include=['float64', 'int64'])

scaler = StandardScaler()
numerical_df_scaled = scaler.fit_transform(numerical_df)

pca = PCA(n_components=2)
pca_result = pca.fit_transform(numerical_df_scaled)

df['PC1'] = pca_result[:, 0]
df['PC2'] = pca_result[:, 1]

pca_df = pd.DataFrame(pca_result, columns=['PC1', 'PC2'])

explained_variance = pca.explained_variance_ratio_
print(f"Wariancja wyjaśniana przez każdą składową: {explained_variance}")
print(f"Łączna wariancja: {sum(explained_variance):.2f}")

plt.figure(figsize=(8, 6))
plt.scatter(pca_df['PC1'], pca_df['PC2'], alpha=0.7)
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.title('Wizualizacja danych GPU po PCA')
plt.show()
```

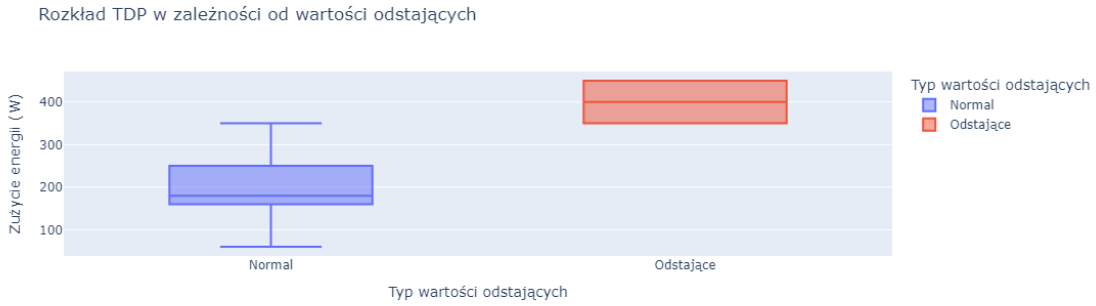
```
Wariancja wyjaśniana przez każdą składową: [0.6000948  0.24329501]
Łączna wariancja: 0.84
```



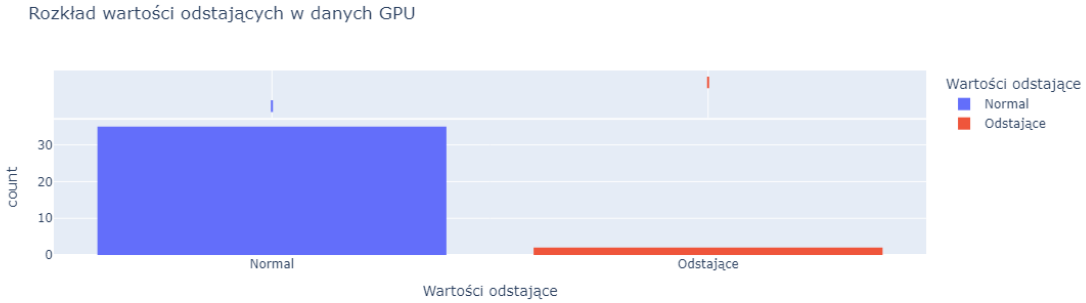
[7]:
```
# 3. Tworzyć zaawansowane interaktywne wizualizacje danych,
df['Różnica między Base a Boost Clock'] = df['Base Clock (MHz)'] - df['Boost␣
 ↪Clock (MHz)']

# Tworzenie wykresu rozrzutu z interaktywnymi danymi
fig_pca = px.scatter(df,
                     x='Różnica między Base a Boost Clock',
                     y='CUDA Cores',
                     color='Memory Size (GB)',
                     hover_data=['Model', 'Base Clock (MHz)', 'Boost Clock␣
 ↪(MHz)', 'CUDA Cores', 'Memory Size (GB)', 'TDP (W)'],
                     title='Zależność między różnicą Base/Boost Clock a ilością␣
 ↪rdzeni CUDA',
                     labels={'Różnica między Base a Boost Clock': 'Różnica Base/
 ↪Boost Clock (MHz)',
                             'CUDA Cores': 'Liczba rdzeni CUDA',
                             'Memory Size (GB)': 'Pamięć (GB)'})
fig_pca.show()
```



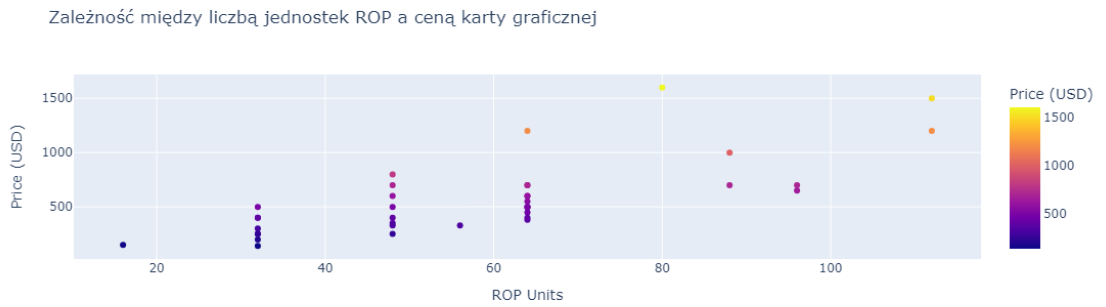
[9]:
```
fig_box = px.box(df,
                 x='Wartości odstające',
                 y='TDP (W)',  # Możemy analizować np. TDP (W) lub inną␣
 ↪interesującą zmienną
                 color='Wartości odstające',
                 title='Rozkład TDP w zależności od wartości odstających',
                 labels={'TDP (W)': 'Zużycie energii (W)', 'Wartości odstające':
 ↪ 'Typ wartości odstających'})
fig_box.show()
```



[11]:
```
fig_hist = px.histogram(df,
                        x='Wartości odstające',
                        color='Wartości odstające',
                        title='Rozkład wartości odstających w danych GPU',
                        marginal='box')  # Dodanie wykresu pudełkowego na␣
 ↪marginesie

fig_hist.show()
```



[13]:
```
fig_children_income = px.scatter(df,
                                 x='ROP Units',
                                 y='Price (USD)',
                                 color='Price (USD)',
                                 hover_data=['Model', 'ROP Units', 'Price␣
 ↪(USD)', 'TDP (W)', 'Memory Size (GB)'],
                                 title='Zależność między liczbą jednostek ROP a␣
 ↪ceną karty graficznej')

fig_children_income.show()
```

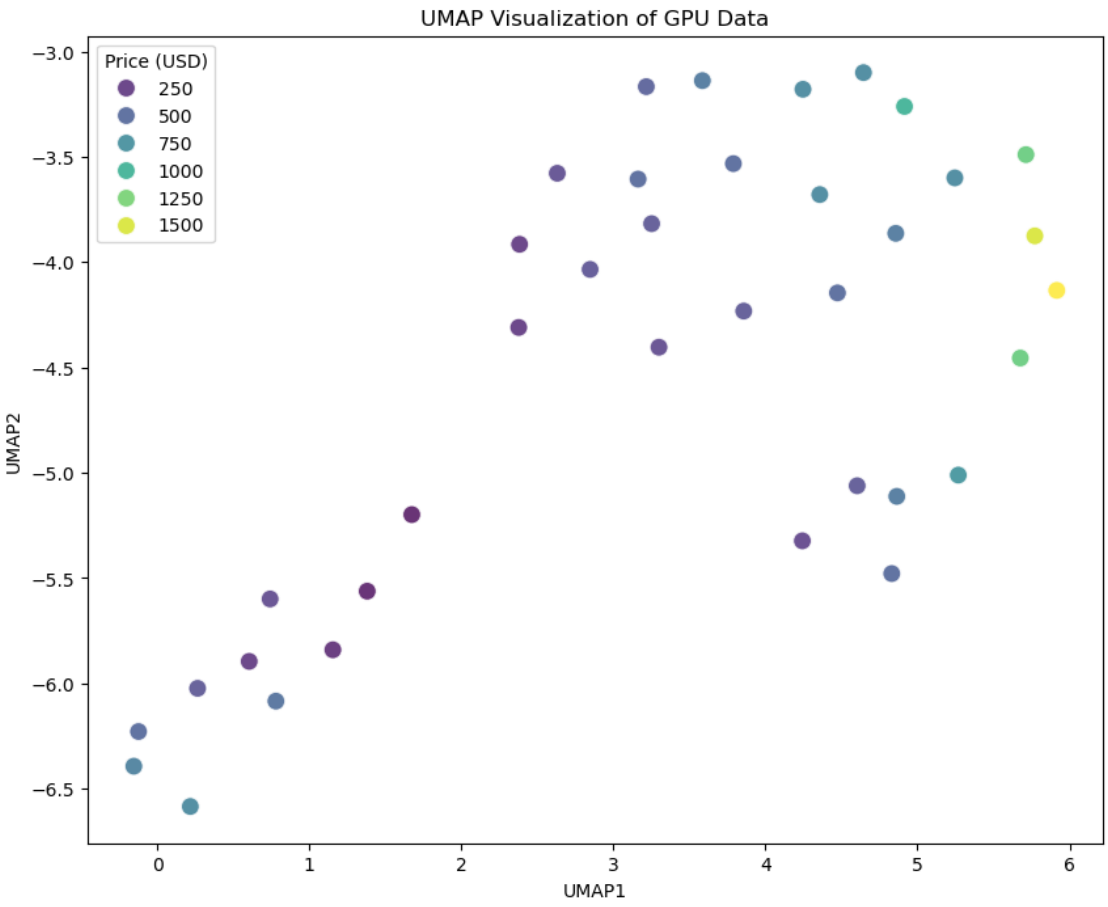


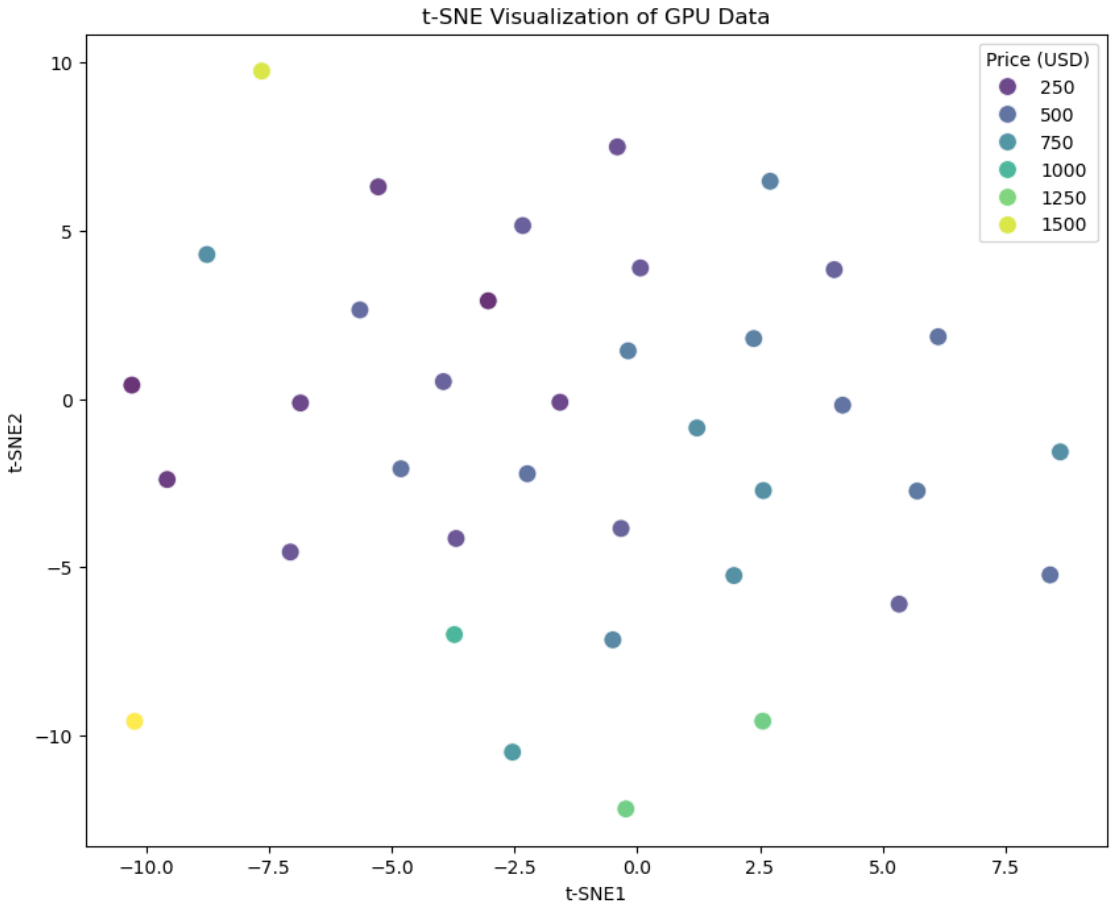
[15]:
```
# 4. Zwizualizować dane wielowymiarowe za pomocą zaawansowanych algorytmów␣
 ↪(t-SNE, UMAP)
!pip install umap-learn
from sklearn.manifold import TSNE
from umap import UMAP


# Wybór interesujących kolumn
```

```
columns_of_interest = ['TDP (W)', 'Price (USD)', 'CUDA Cores', 'Memory Size␣
 ↪(GB)', 'Base Clock (MHz)', 'Boost Clock (MHz)']
X = df[columns_of_interest]

# Przeskalowanie danych
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Użycie UMAP do redukcji wymiarowości
umap_model = UMAP(n_components=2, random_state=42)
umap_results = umap_model.fit_transform(X_scaled)

# Użycie t-SNE do redukcji wymiarowości
tsne_model = TSNE(n_components=2, random_state=42, perplexity=30, n_iter=300)
tsne_results = tsne_model.fit_transform(X_scaled)

# Przekształcenie wyników do DataFrame
umap_df = pd.DataFrame(umap_results, columns=['UMAP1', 'UMAP2'])
tsne_df = pd.DataFrame(tsne_results, columns=['t-SNE1', 't-SNE2'])

# Dodanie wyników do oryginalnego DataFrame
df_umap = pd.concat([df, umap_df], axis=1)
df_tsne = pd.concat([df, tsne_df], axis=1)

# Wizualizacja UMAP
plt.figure(figsize=(10, 8))
sns.scatterplot(data=df_umap, x='UMAP1', y='UMAP2', hue='Price (USD)',␣
 ↪palette='viridis', s=100, alpha=0.8)
plt.title('UMAP Visualization of GPU Data')
plt.show()

# Wizualizacja t-SNE
plt.figure(figsize=(10, 8))
sns.scatterplot(data=df_tsne, x='t-SNE1', y='t-SNE2', hue='Price (USD)',␣
 ↪palette='viridis', s=100, alpha=0.8)
plt.title('t-SNE Visualization of GPU Data')
plt.show()
```

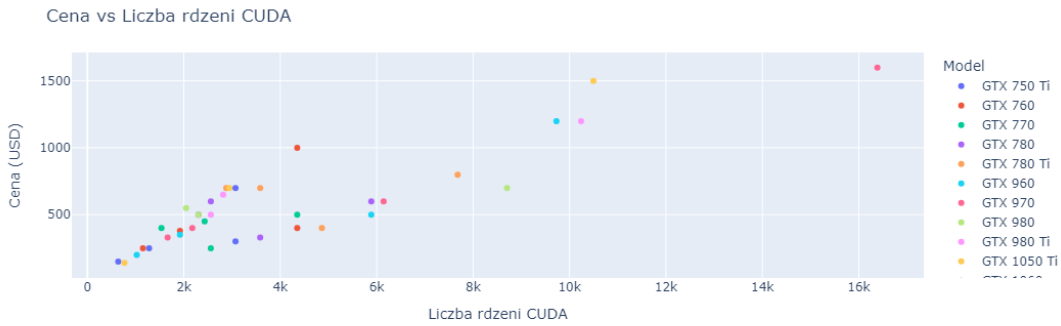
```
Requirement already satisfied: umap-learn in
c:\hubert\programy\anaconda\lib\site-packages (0.5.7)
Requirement already satisfied: numpy>=1.17 in
c:\hubert\programy\anaconda\lib\site-packages (from umap-learn) (1.26.4)
Requirement already satisfied: scipy>=1.3.1 in
c:\hubert\programy\anaconda\lib\site-packages (from umap-learn) (1.13.1)
Requirement already satisfied: scikit-learn>=0.22 in
c:\hubert\programy\anaconda\lib\site-packages (from umap-learn) (1.4.2)
Requirement already satisfied: numba>=0.51.2 in
```

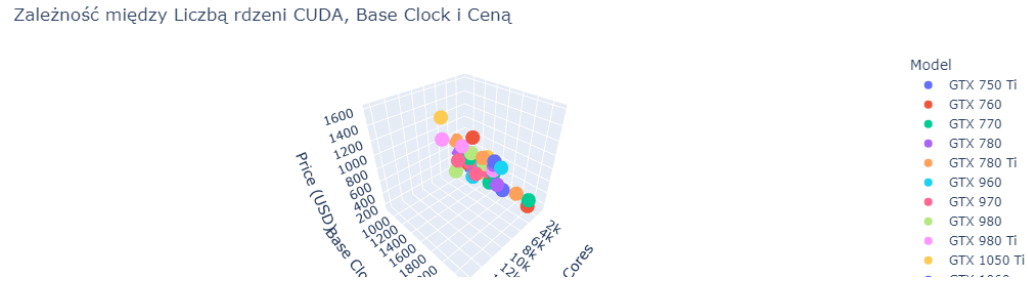
```
c:\hubert\programy\anaconda\lib\site-packages (from umap-learn) (0.59.1)
Requirement already satisfied: pynndescent>=0.5 in
c:\hubert\programy\anaconda\lib\site-packages (from umap-learn) (0.5.13)
Requirement already satisfied: tqdm in c:\hubert\programy\anaconda\lib\site-
packages (from umap-learn) (4.66.4)
Requirement already satisfied: llvmlite<0.43,>=0.42.0dev0 in
c:\hubert\programy\anaconda\lib\site-packages (from numba>=0.51.2->umap-learn)
(0.42.0)
Requirement already satisfied: joblib>=0.11 in
c:\hubert\programy\anaconda\lib\site-packages (from pynndescent>=0.5->umap-
learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in
c:\hubert\programy\anaconda\lib\site-packages (from scikit-learn>=0.22->umap-
learn) (2.2.0)
Requirement already satisfied: colorama in c:\hubert\programy\anaconda\lib\site-
packages (from tqdm->umap-learn) (0.4.6)

C:\Hubert\Programy\anaconda\Lib\site-packages\umap\umap_.py:1952: UserWarning:

n_jobs value 1 overridden to 1 by setting random_state. Use no seed for
parallelism.
```





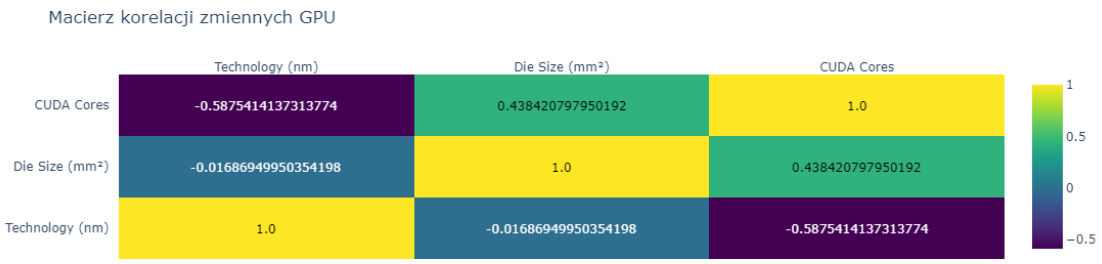
[16]:
```
# 5. tworzyć interaktywne wizualizacje danych w 2D i 3D,
# Wykres 2D: Cena vs CUDA Cores
fig = px.scatter(df, x='CUDA Cores', y='Price (USD)', color='Model',
                 hover_data=['Model', 'Price (USD)', 'CUDA Cores'])
fig.update_layout(title="Cena vs Liczba rdzeni CUDA",
                  xaxis_title="Liczba rdzeni CUDA",
                  yaxis_title="Cena (USD)")
fig.show()
fig = px.scatter_3d(df, x='CUDA Cores', y='Base Clock (MHz)', z='Price (USD)',
                    color='Model', hover_data=['Model', 'CUDA Cores', 'Base␣
 ↪Clock (MHz)', 'Price (USD)'])
fig.update_layout(title="Zależność między Liczbą rdzeni CUDA, Base Clock i␣
 ↪Ceną")
fig.show()
```





[18]:
```python
# 6. Analizować zależności między zmiennymi za pomocą macierzy korelacji.
import plotly.figure_factory as ff

numerical_features = [
    'Technology (nm)', 'Die Size (mm²)','CUDA Cores'
]
numeric_df = df[numerical_features]

# Tworzenie macierzy korelacji
correlation_matrix = numeric_df.corr()

# Wizualizacja macierzy korelacji za pomocą heatmapy
fig = ff.create_annotated_heatmap(
    z=correlation_matrix.values,
    x=correlation_matrix.columns.tolist(),
    y=correlation_matrix.columns.tolist(),
    colorscale='Viridis',
    showscale=True,
)
```

```
# Aktualizacja tytułu wykresu
fig.update_layout(title="Macierz korelacji zmiennych GPU")

# Wyświetlenie wykresu
fig.show()
```



[21]:
```
# 7. Przeprowadzać testy statystyczne dla analizy różnic w grupach.
from scipy import stats

# Tworzenie grup "GTX" i "RTX" na podstawie nazwy modelu
gtx_group = df[df['Model'].str.contains('GTX')]['Price (USD)']
rtx_group = df[df['Model'].str.contains('RTX')]['Price (USD)']

# Test T-studenta dla różnic w cenie
t_stat, p_value = stats.ttest_ind(gtx_group, rtx_group)

print(f"T-statystyka: {t_stat}")
print(f"P-wartość: {p_value}")
if p_value < 0.05:
    print("Istnieje istotna różnica w średnich cenach między kartami GTX a RTX.
 ↪")
else:
    print("Brak istotnej różnicy w średnich cenach między kartami GTX a RTX.")
```

```
T-statystyka: -2.464907742937319
P-wartość: 0.018756532759691073
Istnieje istotna różnica w średnich cenach między kartami GTX a RTX.
```

[23]:
```
from statsmodels.formula.api import ols
from statsmodels.stats.anova import anova_lm

# Utworzenie nowej kolumny 'Technology' dla GTX/RTX
df['Technology'] = df['Model'].apply(lambda x: 'GTX' if 'GTX' in x else 'RTX')
```

```
df.rename(columns={'Price (USD)': 'Price'}, inplace=True)
# Model ANOVA dla analizy ceny w zależności od technologii (GTX vs RTX)
model = ols('Price ~ C(Technology)', data=df).fit()
anova_table = anova_lm(model)

print(anova_table)

if anova_table['PR(>F)'].iloc[0] < 0.05:
    print("Istnieje istotna różnica w średnich cenach między kartami GTX a RTX.
 ↪")
else:
    print("Brak istotnej różnicy w średnich cenach między kartami GTX a RTX.")
```

```
                df        sum_sq        mean_sq        F    PR(>F)
C(Technology)  1.0  6.335137e+05  633513.718264  6.07577  0.018757
Residual      35.0  3.649411e+06  104268.874459      NaN       NaN
Istnieje istotna różnica w średnich cenach między kartami GTX a RTX.
```

[ ]: