Eksplorowanie Big Data za pomocą Pythona

W tej części zajmiemy się niektórymi narzędziami i procesami używanymi przez analityków danych do formatowania, przetwarzania i wysyłania zapytań do danych. Dostępnych jest wiele narzędzi i bibliotek opartych na języku Python (takich jak „R"), ale zdecydowaliśmy się użyć NumPy z trzech powodów. Po pierwsze, jest to jedno z dwóch najpopularniejszych narzędzi do nauki o danych w Pythonie. Po drugie, wiele projektów zorientowanych na sztuczną inteligencję używa NumPy (takich jak ten z naszego ostatniego rozdziału). Po trzecie, bardzo przydatny pakiet nauki o danych w języku Python, Pandas, jest oparty na NumPy. Pandas okazuje się być bardzo ważnym pakietem w data science. Sposób hermetyzacji danych w bardziej abstrakcyjny sposób ułatwia manipulowanie, dokumentowanie i zrozumienie przekształceń dokonywanych w podstawowych zestawach danych. Wreszcie MatPlotLib to dobry pakiet do wizualizacji wyników big data. Jest bardzo zorientowany na Pythona, ale wymaga stromej krzywej uczenia się, aby zacząć. Zostało to jednak do pewnego stopnia poprawione dzięki nowym pakietom dodatkowym, takim jak „morski". Podsumowując, są to rozsądne pakiety do zaatakowania problemu nauki o danych i uzyskania wyników wprowadzających w tę interesującą dziedzinę.

Przedstawiamy NumPy, Pandas i MatPlotLib

Za każdym razem, gdy spojrzysz na społeczności zajmujące się komputerami naukowymi i nauką o danych, ciągle pojawiają się trzy kluczowe pakiety Pythona:

* NumPy

* Pandas

* MatPlotLib

Zostały one omówione w kilku następnych sekcjach.

NumPy

NumPy dodaje do Pythona narzędzia do manipulacji dużymi danymi, takie jak manipulowanie dużymi tablicami i funkcje matematyczne wysokiego poziomu do nauki o danych. NumPy najlepiej radzi sobie z podstawowymi obliczeniami numerycznymi, takimi jak średnie, średnie i tak dalej. Doskonale sprawdza się również w tworzeniu i manipulowaniu wielowymiarowymi tablicami zwanymi tensorami lub macierzami. W księdze 4 intensywnie używaliśmy NumPy do manipulowania danymi i tensorami w sieciach neuronowych i uczeniu maszynowym. Jest to wyjątkowe narzędzie do zastosowań sztucznej inteligencji. Istnieje wiele dobrych samouczków dla NumPy w Internecie. Wybór niektórych dobrych krok po kroku to:

* NumPy Tutorial Part 1 – Wprowadzenie do tablic

(https://www.machinelearningplus.com/python/numpy-tutorial-part1-array-pythonexamples/) : Dobre wprowadzenie do macierzy (znanych również jako tensory) i ich dopasowania do LiczbaPy.

* Samouczek NumPy (https://www.ttorialspoint.com/numpy): Ładny przegląd NumPy, skąd pochodzi i jak go używać.

* Samouczek NumPy: ucz się na przykładzie (https://www.guru99.com/numpy-tutorial.html): Mniej teorii, ale mnóstwo świetnych przykładów uzupełnij praktyczne luki po zapoznaniu się z dwoma pierwszymi samouczkami.

Oto prosty przykład programu NumPy. Ten program buduje macierz 2x2, a następnie wykonuje różne operacje zorientowane na macierz na macierzy:

```
import numpy as np

x = np.array([[1,2],[3,4]])

print(np.sum(x)) # Compute sum of all elements; prints "10"

print(np.sum(x, axis=0)) # Compute sum of each column; prints "[4 6]"

print(np.sum(x, axis=1)) # Compute sum of each row; prints "[3 7]"
```

Pandas

Python doskonale nadaje się do przetwarzania i przygotowywania danych, ale nie nadaje się do analizy i modelowania danych. Pandas wypełnia tę lukę. Pandas zapewnia szybkie, elastyczne i wyraziste struktury danych, dzięki którym praca z danymi relacyjnymi lub danymi z etykietami jest bardziej intuicyjna. Naszym zdaniem jest to podstawowy budulec do analizy danych w świecie rzeczywistym w Pythonie. Działa dobrze z danymi tabelarycznymi (takimi jak tabele SQL lub arkusze kalkulacyjne Excel) i jest naprawdę dobry z danymi szeregów czasowych (takich jak, powiedzmy, temperatury mierzone co godzinę). Pamiętasz naszą dyskusję na temat masowania danych? Radzenie sobie z brakującymi lub złymi danymi? Jest to jedna z rzeczy, do których Pandas jest przeznaczony i robi to naprawdę dobrze. Pozwala również na tworzenie złożonych, hierarchicznych struktur danych, do których można uzyskać dostęp za pomocą funkcji Pandy w bardzo intuicyjny sposób. Możesz scalać i łączyć zestawy danych, a także konwertować wiele typów danych na wszechobecne obiekty danych Pandas, DataFrames. Pandas jest oparty na NumPy i ma taką samą szybkość jak ta biblioteka Pythona, i może osiągnąć duży wzrost szybkości w porównaniu z prostym kodem Pythona obejmującym pętle. Pandas DataFrames to sposób przechowywania danych w prostokątnych siatkach, które można łatwo przeglądać. DataFrame może zawierać inne DataFrame, jednowymiarowe serie danych, tensor NumPy (tablica — znowu zaczynamy z podobieństwami do części w sieciach neuronowych i uczeniu maszynowym) oraz słowniki tensorów i macierzy. Oprócz danych możesz także określić indeksy i nazwy kolumn dla ramki DataFrame. Dzięki temu kod jest bardziej zrozumiały do analizy danych i manipulowania nimi. Możesz uzyskiwać dostęp, usuwać i zmieniać nazwy komponentów DataFrame w miarę wprowadzania większej liczby struktur i dołączania większej liczby powiązanych danych do swojej struktury DataFrame.

MatPlotLib

MatPlotLib to biblioteka, która dodaje do Pythona brakujące funkcje wizualizacji danych. Został zaprojektowany jako uzupełnienie wykorzystania NumPy w analizie danych i programach naukowych. Zapewnia obiektowy interfejs API języka Python (interfejs programowania aplikacji) do osadzania wykresów w aplikacjach przy użyciu interfejsów GUI ogólnego przeznaczenia. Dla osób zaznajomionych z MatLab, MatPlotLib udostępnia wersję proceduralną o nazwie PyLab. Dzięki MatPlotLib możesz tworzyć rozbudowane i profesjonalnie wyglądające wykresy, a nawet tworzyć wykresy „na żywo", które aktualizują się podczas działania aplikacji. Może to być przydatne w aplikacjach do uczenia maszynowego i aplikacjach do analizy danych, gdzie dobrze jest widzieć, jak system robi postępy w osiąganiu jakiegoś celu.

Realizowanie pierwszego projektu związanego z nauką o danych

Czas, abyśmy zmusili NumPy i Pandas do pracy nad prostym projektem analizy danych. Zamierzam wybrać nasz zestaw danych ze strony internetowej Kaggle.com. Kaggle, którego mottem jest „Your

Home for Data Science", to należąca do Google internetowa społeczność naukowców zajmujących się danymi i użytkowników. W większości przypadków Kaggle pozwala użytkownikom znajdować zestawy danych, pobierać dane i wykorzystywać je na bardzo otwartych licencjach. Kaggle obsługuje również solidny zestaw konkursów dotyczących rozwiązywania problemów związanych z uczeniem maszynowym, często ogłaszanych przez firmy, które naprawdę potrzebują rozwiązania. W przypadku tego pierwszego problemu chcę wybrać całkiem prosty zestaw danych. Diamenty są najlepszym przyjacielem analityka danych. Wybrałem bazę danych „diamenty" z Kaggle.com, ponieważ ma dość prostą strukturę i zawiera tylko około 54 000 elementów — jest łatwa w użyciu dla naszego komputera Raspberry Pi. Można go pobrać ze strony https://www.kaggle.com/shivam2503/diamonds. Korzystanie z Kaggle wymaga zarejestrowania się i zalogowania do społeczności, ale nie wiąże się to z żadnymi kosztami. Metadane (metadane to dane opisujące dane, stąd metadane) składają się z dziesięciu zmiennych, które również można traktować jako nagłówki kolumn.

Nagłówek kolumny : Typ danych : Opis

Licznik indeksów : Numeryczny

carat : Liczbowy : Masa diamentu w karatach

cięcie :Tekst : Opisz jakość szlifu diamentu. Jakość w kolejności rosnącej Dostateczna, dobra, bardzo dobra, premium, idealna

color : Text : Kolor diamentu, gdzie D oznacza najlepszy, a J najgorszy

przejrzystość :Tekst: Jak oczywiste są inkluzje w diamentie: (w kolejności od najlepszej do najgorszej, FL = bez skazy, I3 = inkluzje poziomu 3) FL,IF, VVS1, VVS2, VS1, VS2, SI1, SI2, I1, I2, I3

głębokość : Numeryczne : Głębokość %: Wysokość diamentu mierzona od kuli do stołu, podzielona przez średnią średnicę obręczy

table : Numeric : Table %: Szerokość stołu diamentu wyrażona jako procent jego średniej średnicy

price : Numeric : Cena diamentu

x : Liczbowe : Długość mm

y : Numeryczne : Szerokość mm

x : Numeryczne : Głębokość mm

Gdybyś miał użyć tego jako zestawu szkoleniowego dla programu uczenia maszynowego, zobaczyłbyś program wykorzystujący NumPy i TensorFlow bardzo podobny do tego, który pokazaliśmy wcześniej. Tu pokażemy Ci zestaw prostej analizy danych opartej na pandach, aby odczytać nasze dane i zadać kilka pytań. Zamierzam użyć DataFrame w pandach (struktura danych z etykietami 2D z kolumnami, które mogą należeć do różnych typów). Struktura danych Panele to trójwymiarowy kontener danych. W tym przykładzie trzymam się DataFrames, ponieważ DataFrames ułatwia wizualizację danych 2D. Jeśli instalujesz NumPy i pandy na Raspberry Pi, użyj tych poleceń:

sudo apt-get install python3-numpy

sudo apt-get install python3-pandas

Teraz czas na przykład.

Używając nano (lub swojego ulubionego edytora tekstu), otwórz plik o nazwie FirstDiamonds.py i wprowadź następujący kod:

```
# Diamonds are a Data Scientist's Best Friend
#import the pandas and numpy library
import numpy as np
import pandas as pd
# read the diamonds CSV file
# build a DataFrame from the data
df = pd.read_csv('diamonds.csv')
print (df.head(10))
print()
# calculate total value of diamonds
sum = df.price.sum()
print ("Total $ Value of Diamonds: ${:0,.2f}".format( sum))
# calculate mean price of diamonds
mean = df.price.mean()
print ("Mean $ Value of Diamonds: ${:0,.2f}".format(mean))
# summarize the data
descrip = df.carat.describe()
print()
print (descrip)
descrip = df.describe(include='object')
print()
print (descrip)
```

Upewniając się, że w swoim katalogu znajduje się plik diamonds.csv, uruchom następujące polecenie:

```
python3 FirstDiamonds.py
```

Powinieneś zobaczyć następujące wyniki:

```
   Unnamed: 0  carat        cut color clarity  depth  table  price     x
   y     z
0           1   0.23      Ideal     E     SI2   61.5   55.0    326  3.95
   3.98  2.43
1           2   0.21    Premium     E     SI1   59.8   61.0    326  3.89
   3.84  2.31
2           3   0.23       Good     E     VS1   56.9   65.0    327  4.05
   4.07  2.31
3           4   0.29    Premium     I     VS2   62.4   58.0    334  4.20
   4.23  2.63
4           5   0.31       Good     J     SI2   63.3   58.0    335  4.34
   4.35  2.75
5           6   0.24  Very Good     J    VVS2   62.8   57.0    336  3.94
   3.96  2.48
6           7   0.24  Very Good     I    VVS1   62.3   57.0    336  3.95
   3.98  2.47
```

```
7           8   0.26  Very Good     H     SI1   61.9   55.0    337  4.07
   4.11  2.53
8           9   0.22       Fair     E     VS2   65.1   61.0    337  3.87
   3.78  2.49
9          10   0.23  Very Good     H     VS1   59.4   61.0    338  4.00
   4.05  2.39

Total $ Value of Diamonds: $212,135,217.00
Mean $ Value of Diamonds: $3,932.80

count    53940.000000
mean         0.797940
std          0.474011
min          0.200000
25%          0.400000
50%          0.700000
75%          1.040000
max          5.010000
Name: carat, dtype: float64

          cut  color clarity
count   53940  53940   53940
unique      5      7       8
top     Ideal      G     SI1
freq    21551  11292   13065
```

To dużo danych jak na krótki fragment kodu!

Łamanie kodu

Diamonds are a Data Scientist's Best Friend

Najpierw importujemy wszystkie potrzebne biblioteki:

#import the pandas and numpy library

import numpy as np

import pandas as pd

Wczytaj plik diamentów do pandas DataFrame. Uwaga: nie musieliśmy formatować ani modyfikować danych w tym pliku. To nie jest normalna sytuacja w prawdziwej nauce o danych. Często spędzasz znaczną ilość czasu na dostarczaniu danych tam, gdzie chcesz - czasami tyle czasu, co cała reszta projektu.

```
# read the diamonds CSV file

# build a DataFrame from the data

df = pd.read_csv('diamonds.csv')
```

Dla sprawdzenia poprawności wydrukujmy pierwsze dziesięć wierszy w DataFrame.

```
print (df.head(10))

print()
```

Tutaj obliczamy kilka wartości z kolumny o nazwie cena. Zauważ, że możemy użyć kolumny jako części obiektu DataFrame. To wspaniale, że możesz to zrobić za pomocą Pythona!

```
# calculate total value of diamonds

sum = df.price.sum()

print ("Total $ Value of Diamonds: ${:0,.2f}".format( sum))

# calculate mean price of diamonds

mean = df.price.mean()

print ("Mean $ Value of Diamonds: ${:0,.2f}".format(mean))
```

Teraz uruchamiamy wbudowaną funkcję opisującą, aby najpierw opisać i podsumować dane dotyczące karatów.

```
# summarize the data

descrip = df.carat.describe()

print()

print (descrip)
```

Ta następna instrukcja drukuje opis wszystkich kolumn nieliczbowych w naszej ramce DataFrame: w szczególności kolumny cięcia, koloru i przejrzystości:

```
descrip = df.describe(include='object')

print()

print (descrip)
```

Aby zainstalować MatPlotLib na swoim Raspberry Pi, wpisz pip3 install matplotlib.

Wizualizacja danych za pomocą MatPlotLib

Teraz przechodzimy do wizualizacji danych za pomocą MatPlotLib. Używamy MatPlotLib do rysowania wykresów związanych ze sposobem, w jaki nasz program do uczenia maszynowego poprawiał swoją dokładność podczas szkolenia. Teraz używamy MatPlotLib, aby pokazać kilka interesujących rzeczy o naszym zbiorze danych. Aby te programy działały, musisz je uruchomić z okna terminala w GUI Raspberry Pi. Możesz użyć VNC, aby uzyskać GUI, jeśli używasz Raspberry Pi bez głowy. Jedną z naprawdę przydatnych rzeczy w pandach i MatPlotLib jest to, że typy NumPy i DataFrame są bardzo kompatybilne z wymaganymi formatami graficznymi. Wszystkie oparte są na macierzach i tablicach

NumPy. Nasz pierwszy wykres to wykres punktowy przedstawiający czystość diamentu w funkcji jego wielkości w karatach.

Czystość diamentu a rozmiar w karatach

Używając nano (lub swojego ulubionego edytora tekstu), otwórz plik o nazwie Plot_Clarity VSCarat.py i wprowadź następujący kod:

```
# Looking at the Shiny Diamonds

#import the pandas and numpy library

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

# read the diamonds CSV file

# build a DataFrame from the data

df = pd.read_csv('diamonds.csv')

import matplotlib.pyplot as plt

carat = df.carat

clarity = df.clarity

plt.scatter(clarity, carat)

plt.show() # or plt.savefig("name.png")
```

Uruchom swój program. Jak to się ma do łatwości w spiskowaniu? Pandy i MatPlotLib idą ramię w ramię. Pamiętaj, że czystość diamentu mierzy się na podstawie widocznych inkluzji w diamencie: FL, IF, VVS1, VVS2, VS1, VS2, SI1, SI2, I1, I2, I3 (w kolejności od najlepszego do najgorszego: FL = bezszwowy, I3 = inkluzje poziomu 3). Zwróć uwagę, że w naszej bazie danych diamentów nie było diamentów bez piły. Można by pokusić się o stwierdzenie, że największe diamenty są oceniane jako IF. Pamiętaj jednak, że tak naprawdę nie masz pojęcia, w jaki sposób zebrano te dane więc naprawdę nie można wyciągać tak ogólnych wniosków. Wszystko, co możesz powiedzieć, to to, że „W tym zbiorze danych przejrzystość„ IL ”ma największe diamenty”.

Liczba diamentów w każdym rodzaju czystości

Używając nano (lub swojego ulubionego edytora tekstu), otwórz plik o nazwie Plot_Count Clarity.py i wprowadź następujący kod:

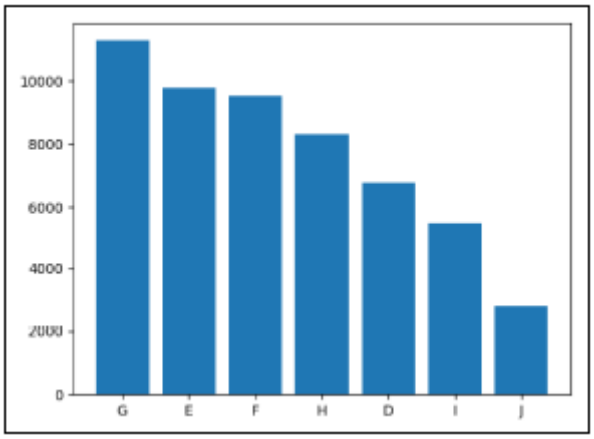
```
# Looking at the Shiny Diamonds

#import the pandas and numpy library

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

# read the diamonds CSV file
```

```
# build a DataFrame from the data

df = pd.read_csv('diamonds.csv')

import matplotlib.pyplot as plt

# count the number of each textual type of clarity

clarityindexes = df['clarity'].value_counts().index.tolist()

claritycount= df['clarity'].value_counts().values.tolist()

print(clarityindexes)

print(claritycount)

plt.bar(clarityindexes, claritycount)

plt.show() # or plt.savefig("name.png")
```

Ponownie pamiętaj, że czystość diamentu jest mierzona przez to, jak oczywiste są inkluzje w diamencie: FL,IF, VVS1, VVS2, VS1, VS2, SI1, SI2, I1, I2, I3 (w kolejności od najlepszego do najgorszego: FL = bez skazy, I3 = inkluzje poziomu 3). Zwróć uwagę, że w naszej bazie diamentów nie mieliśmy diamentów bez skazy. Na tym wykresie widać, że diamenty średniej jakości SI1, VS2 i SI2 są najczęściej reprezentowane w naszym zbiorze danych dotyczących diamentów. Liczba diamentów w każdym typie koloru Przyjrzałem się przejrzystości, teraz spójrzmy na rodzaj koloru w naszym stosie diamentów. Używając nano (lub swojego ulubionego edytora tekstu), otwórz plik o nazwie Plot_CountColor.py i wprowadź następujący kod

```
# Looking at the Shiny Diamonds

#import the pandas and numpy library

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

# read the diamonds CSV file

# build a DataFrame from the data

df = pd.read_csv('diamonds.csv')

import matplotlib.pyplot as plt

# count the number of each textual type of color

colorindexes = df['color'].value_counts().index.tolist()

colorcount= df['color'].value_counts().values.tolist()

print(colorindexes)

print(colorcount)

plt.bar(colorindexes, colorcount)

plt.show() # or plt.savefig("name.png")
```

Uruchom swój program.



Kolor „G” reprezentuje około 25 procent wielkości naszej próby. To „G” jest prawie bezbarwne. Ogólna zasada jest taka, że mniej koloru, wyższa cena. Wyjątkiem są róże i błękity, które znajdują się poza tym odwzorowaniem kolorów i próbką. Używanie Pand do znajdowania korelacji: wykresy ciepła Ostatni wykres, który chcę wam pokazać, nazywa się wykresem ciepła. Służy do graficznego przedstawiania korelacji między wartościami liczbowymi w naszej bazie danych. Na tym wykresie bierzemy wszystkie wartości liczbowe i tworzymy macierz korelacji, która pokazuje, jak bardzo są one ze sobą skorelowane. Aby szybko i łatwo wygenerować ten wykres, używamy innej biblioteki dla Pythona i MatPlotLib o nazwie seaborn. Seaborn zapewnia API zbudowane na bazie MatPlotLib, które integruje się z pandas DataFrames, co czyni go idealnym do nauki o danych. Jeśli nie masz jeszcze seaborn na swoim Raspberry Pi (a jeśli zainstalowałeś MatPlotLib, prawdopodobnie już to robisz). Uruchom przykładowy program w języku Python Plot_Heat.py, aby sprawdzić, czy tak jest. Jeśli nie, uruchom następujące polecenie:

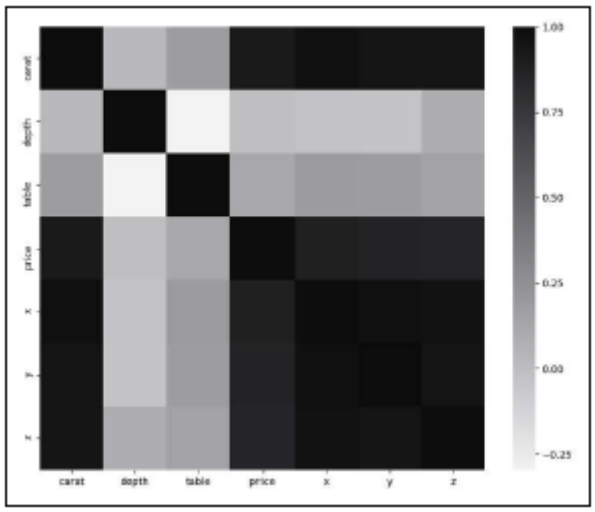
sudo apt-get install python3-seaborn

Używając nano (lub swojego ulubionego edytora tekstu), otwórz plik o nazwie Plot_Heat.py i wprowadź następujący kod:

```
# Looking at the Shiny Diamonds

#import the pandas and numpy library

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

# read the diamonds CSV file

# build a DataFrame from the data

df = pd.read_csv('diamonds.csv')

# drop the index column

df = df.drop('Unnamed: 0', axis=1)
```

```
f, ax = plt.subplots(figsize=(10, 8))

corr = df.corr()

print (corr)

sns.heatmap(corr, mask=np.zeros_like(corr, dtype=np.bool),

cmap=sns.diverging_palette(220, 10, as_cmap=True),

square=True, ax=ax)

plt.show()
```

Uruchom program i rozkoszuj się wizualizacją danych rzeczywistych na rysunku 2.4. Pierwszą rzeczą, na którą należy zwrócić uwagę w przypadku rysunku , jest to, że im bardziej czerwony kolor, tym wyższa korelacja między dwiema zmiennymi.



Ukośny pasek od lewego górnego rogu do dolnego górnego rogu pokazuje, że na przykład karat koreluje w 100 procentach z karatem. Żadnej niespodzianki. Zmienne x, y i z są ze sobą dość skorelowane, co oznacza, że gdy diamenty w naszej bazie danych rosną w jednym wymiarze, rosną również w pozostałych dwóch wymiarach. A co z ceną? Wraz ze wzrostem karatów i wielkości rośnie cena. To ma sens. Co ciekawe, głębokość (wysokość diamentu mierzona od kulki do stołu, podzielona przez średnią średnicę obręczy) wcale nie jest silnie skorelowana z ceną, aw rzeczywistości jest nieco ujemnie skorelowana. To niesamowite, ile wniosków można wyciągnąć z tego rodzaju mapy. Mapy cieplne świetnie nadają się do wykrywania ogólnych korelacji krzyżowych w danych. Byłoby interesujące zobaczyć korelację między kolorem/przezroczystością a ceną. Dlaczego nie ma tego na tym wykresie? Dzieje się tak, ponieważ te kolumny są tekstowe i można wykonywać tylko korelacje na wartościach liczbowych. Jak możesz to naprawić? Zastępując każdą literę koloru kodem numerycznym (na przykład 1–8), a następnie ponownie generując wykres cieplny. Tę samą technikę można zastosować w przypadku czystości diamentu.