

POLITECHNIKA WROCŁAWSKA
WYDZIAŁ ELEKTRONIKI

KIERUNEK: INFORMATYKA
SPECJALNOŚĆ: SYSTEMY I SIECI KOMPUTEROWE

PRACA DYPLOMOWA
INŻYNIERSKA

Wizualizacja danych geograficznych z otwartych
źródeł z wykorzystaniem popularnych
frameworków języka Python

Visualization of geographical open source data
utilizing popular Python frameworks

AUTOR:

Łukasz Kowalik

PROWADZĄCY PRACĘ:
dr inż. Piotr Sobolewski, K32Wo4Do3

Spis treści

1. Wstęp	7
2. Wprowadzenie do analizy geoprzestrzennej	8
2.1. Pojęcie geoprzestrzeni	8
2.2. Znaczenie analizy geoprzestrzennej	8
2.3. Kluczowe aspekty teoretyczne systemów GIS	9
2.3.1. Lokalizacja	9
2.3.2. Odwzorowania geograficzne	9
2.3.3. Dane wektorowe	10
2.3.4. Wybrane typy wizualizacji kartograficznych	13
3. Przegląd i analiza źródeł danych	15
3.1. Geometria granic	15
3.2. Produktu krajowy brutto per capita wyrażony w USD (lata 1960-2019)	16
3.3. Pandemia wirusa COVID-19 na świecie (stan obecny)	17
3.4. Przestępcość w San Francisco (ostatnie siedem dni)	18
3.5. Wypadki drogowe w Wielkiej Brytanii (rok 2015)	20
4. Implementacja projektu	21
4.1. Struktura projektu	21
4.2. Skrypty generujące wizualizacje	22
4.2.1. Najważniejsze wykorzystane biblioteki	22
4.2.2. Schemat działania skryptów generujących wizualizacje	23
4.2.3. Legenda mapy	23
4.2.4. Wizualizacja PKB per capita (lata 1960-2019)	25
4.2.5. Wizualizacja obecnego stanu epidemii COVID-19	29
4.2.6. Wizualizacja najnowszych przestępstw w San Francisco	34
4.2.7. Wizualizacja wypadków drogowych w Wielkiej Brytanii (rok 2015)	36
4.3. Serwer webowy	39
4.3.1. Najważniejsze wykorzystane biblioteki	39
4.3.2. Funkcje aktualizacji danych	39
4.3.3. Zasadniczy kod serwera	41
5. Podsumowanie	44
5.1. Osiągnięcia projektu	44
5.2. Kierunki dalszego rozwoju projektu	44
Literatura	46

Spis rysunków

2.1. Przykład cylindrycznej projekcji Ziemi [21]	9
2.2. Porównanie rzutowań: (a) cylindrycznego, (b) stożkowego, (c) azymutalnego [22]	10
2.3. Przykład kartogramu [14]	13
2.4. Mapa termiczna na przykładzie pamiętnego meczu Brazylia - Niemcy, półfinał mundialu 2014. Bardziej zabawna niż rzetelna (aczkolwiek dobrze obrazująca istotę map termicznych) wizualizacja czasu posiadania piłki w danych obszarach boiska przez obie drużyny [29]	14
2.5. Przykład klastra znaczników na mapie Nowego Jorku [28]	14
3.1. Zrzut ekranu przedstawiający fragment pliku zawierającego dane o PKB per capita	17
3.2. Zrzut ekranu przedstawiający fragment pliku zawierającego dane o epidemii wirusa COVID-19	18
3.3. Struktura zbioru danych dot. przestępcości w San Francisco, [19] [dostęp dnia 03.12.2020]	19
3.4. Zrzut ekranu przedstawiający fragment pliku zawierającego dane o wypadkach drogowych w Wielkiej Brytanii w roku 2015	20
4.1. Zrzut ekranu przedstawiający hierarchię plików wchodzących w skład projektu inżynierskiego	21
4.2. Zestawienie legend z danymi w różnej skali: (a) pięć kategorii danych, (b) jedna kategoria danych, (c) brak danych	25
4.3. Zestawienie rezultatów wizualizacji dla lat: (a) 1962, (b) 2019. Zauważalna jest tendencja do większych braków w danych sprzed kilkudziesięciu lat. Przyczyn tego zjawiska jest wiele, do niektórych z nich można zaliczyć kwestię nieistnienia naówczas niektórych z państw obecnie funkcjonujących, a także brak publikacji tego typu danych ze względów politycznych	28
4.4. Widok wizualizacji dla wybranej warstwy dot. wszystkich potwierdzonych przypadków koronawirusa	33
4.5. Widok wizualizacji dla wybranej warstwy dot. wszystkich potwierdzonych zgonów spowodowanych koronawirusem	33
4.6. Przykład działania wyskakującego okienka z informacją o dokładnej wartości z danej kategorii, w tym przypadku wartość współczynnika przypadków śmiertelnych dla Brazylii	34
4.7. Widok obszaru całego miasta San Francisco oraz poszczególnych zgrupowań znaczników. Najechanie kursorem na symbol zgrupowania skutkuje wyświetleniem kształtu obszaru, który obejmuje	36
4.8. Bardziej szczegółowy widok fragmentu miasta - zgrupowania automatycznie się dzielą i dostosowują do wielkości przybliżenia. Poziom zbliżenia widoczny na zruscie ekranu umożliwia dostrzeżenie co niektórych pojedynczych znaczników, których wybranie skutkuje pojawiением się widocznego "popup'u"	36
4.9. Mapa termiczna wypadków drogowych w Wielkiej Brytanii, 27 września 2015 . .	38

4.10. Mapa termiczna wypadków drogowych w Wielkiej Brytanii, 28 października 2015	38
4.11. Informacje zwrotne z konsoli po uruchomieniu pliku serwerowego <i>app.py</i>	43

Spis listingów

2.1. Przykładowy kod GeoJSON	11
2.2. Przykładowy kod OSM XML	12
3.1. Fragment pliku GeoJSON zawierającego geometrię granic państw świata	15
4.1. Plik legend_template.txt zawierający szablon do tworzenia legendy	23
4.2. Funkcja create_legend z pliku utils.py	24
4.3. Szablon zawartości słownika stylu	26
4.4. Zawartość pliku gdp_viz.py	26
4.5. Funkcja create_covid_viz i jej zależności z pliku covid_viz.py	29
4.6. Funkcja create_sf_crime_viz i jej zależności z pliku sf_crime_viz.py	34
4.7. Funkcja create_uk_accidents_viz i jej zależności z pliku uk_accidents_viz.py	37
4.8. Funkcja download_file i jej zależności z pliku utils.py	39
4.9. Funkcja download_covid_data i jej zależności z pliku covid_viz.py	40
4.10. Funkcja download_sf_crime_data i jej zależności z pliku sf_crime_viz.py	41
4.11. Plik jobs.py	42
4.12. Plik app.py	42

Skróty

API (ang. *Application Programming Interface*)

REST API (ang. *Representational state transfer API*)

HTML (ang. *HyperText Markup Language*)

CSS (ang. *Cascading Style Sheets*)

GIS (ang. *Geographical Information System*)

XML (ang. *Extensible Markup Language*)

JSON (ang. *JavaScript Object Notation*)

OSM (ang. *Open Street Map*)

PKB (*Produkt Krajowy Brutto*)

USD (ang. *United States Dollar*)

GMT (ang. *Greenwich Mean Time*)

PT (ang. *Pacific Time*)

CSV (ang. *Comma-separated Values*)

HTTP (ang. *Hypertext Transfer Protocol*)

URL (ang. *Uniform Resource Locator*)

SoQL (ang. *Socrata Query Language*)

SODA (ang. *Socrata Open Data API*)

Rozdział 1

Wstęp

Celem projektu inżynierskiego jest implementacja wybranych technik wizualizacji ogólnodostępnych, geoprzestrzennych danych z wykorzystaniem otwartego oprogramowania typu API, przeznaczonego do tworzenia map. W ramach jego realizacji wykonano cztery podprojekty wizualizacyjne, spośród których każdy został stworzony z myślą o naświetleniu różnych aspektów związanych z dziedziną analizy geoprzestrzennej:

- celem pierwszego podprojektu jest prezentacja wykorzystania interaktywnego kartogramu zawierającego informacje zmieniające się na przestrzeni czasu,
- rolą drugiego z podprojektów jest przedstawienie sposobu na stworzenie interaktywnego kartogramu opartego o cykliczną i automatyczną aktualizację danych źródłowych,
- trzeci program dotyczy kwestii modelu klastra znaczników i wizualizacji danych opartych o ich "punktowe" położenie geograficzne oraz poszerza temat cyklicznej i automatycznej aktualizacji prezentowanych danych,
- czwarta i ostatnia z wizualizacji dotyczy zagadnienia implementacji interaktywnej mapy cieplnej.

Zakres pracy obejmuje:

- wprowadzenie do kluczowych, w celu zrozumienia treści pracy, pojęć związanych z wizualizacją danych geoprzestrzennych,
- dobór i analizę źródeł danych,
- przedstawienie procesu przetwarzania danych aż do uzyskania gotowych wizualizacji,
- implementacja prostego serwera webowego, jako narzędzia uaktualniającego dane i umożliwiającego powszechny dostęp do stworzonych wizualizacji,
- podsumowanie i spostrzeżenia poczynione w trakcie realizacji projektu inżynierskiego.

W projekcie wykorzystano język programowania Python w wersji 3.8.2 oraz szereg jego darmowych, otwarto-źródłowych modułów, do najważniejszych z których należą *folium* [3] (moduł odpowiadający za komunikację z API usługi OpenStreetMap oraz za tworzenie wizualizacji), *pandas* [6] (analiza, przetwarzanie i manipulacja ustrukturyzowanymi danymi) i *flask* [2] (mikro-framework do tworzenia aplikacji webowych). Ponadto w mniejszym stopniu posłużono się technologiami frontendowymi w postaci języków HTML, CSS oraz JavaScript.

Podczas tworzenia oprogramowania wykorzystano dodatkowe technologie mające na celu wsparcie organizacji pracy programisty. Do zarządzania środowiskiem wirtualnym zawierającym wszelkie potrzebne moduły języka Python wykorzystano menadżer paczek Conda [1], biorąc pod uwagę jego nacisk na wsparcie i dostarczanie narzędzi służących do szeroko pojętej analizy danych. Zarządzanie kodem odbywało się za pośrednictwem systemu kontroli wersji Git [9].

Rozdział 2

Wprowadzenie do analizy geoprzestrzennej

2.1. Pojęcie geoprzestrzeni

Termin *geoprzestrzeń*, którego odmiany zdążyły już wybrzmieć w tekście znaczącą ilość razy, odnosi się do dziedziny odnajdywania informacji ulokowanych na powierzchni Ziemi. Do informacji geoprzestrzennych możemy zaliczyć np. lokalizację stacji badawczej wyrażoną w długości i szerokości geograficznej a także kształt granic danego państwa czy też akwenu wodnego. Żeby pewne dane móc określić mianem geoprzestrzennych musi zajść zatem powiązanie informacji z lokalizacją. Proces uzyskiwania, przetwarzania, a wreszcie wizualizacji takich danych określa się mianem **analizy geoprzestrzennej** [33]. Systemy implementujące powyższy proces nazywamy **Systemami Informacji Geograficznej** (ang. **GIS**).

2.2. Znaczenie analizy geoprzestrzennej

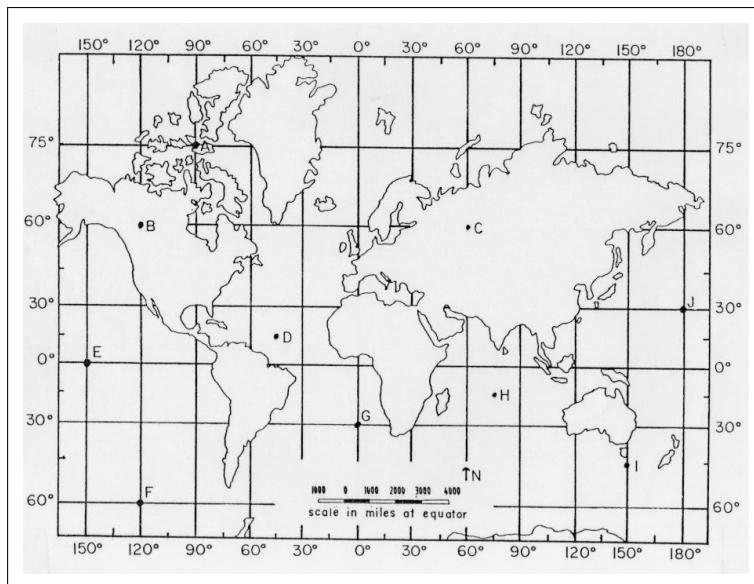
Najważniejszym, sztandarowym celem, jaki ma osiągnąć implementacja rozwiązań geoprzestrzennych jest ułatwienie procesu podejmowania optymalnych decyzji. Nie chodzi tu bynajmniej o podejmowanie ich w zastępstwie za człowieka, a raczej o udzielenie odpowiedzi na krytyczne, nierzadko inaczej niemożliwe do rozwiązania, pytania [32], tym samym przyczyniając się do skutecznej oceny sytuacji. W całości tejże analizy szczególną rolę odgrywają aspekty grafiki oraz wizualizacji. W książce "The Functional Art: An Introduction to Information Graphics and Visualization" Alberto Cairo stawia tezę, iż: "*Grafika nie powinna upraszczać przekazu. Powinna go rozjaśniać, podkreślać tendencje, odkrywać wzorce i rzeczywistość wcześniejsiej nieznaną*" [30]. Cytat ten powinien stanowić kredo dla wszystkich osób zajmujących się tematyką szeroko pojętej wizualizacji danych.

W dobie coraz szybszego rozwoju technologicznego w niespotykanym dotąd tempie rośnie ilość danych, spośród których spora część jest ogólnodostępna i skorzystanie z nich nie wymaga żadnego wysiłku. Gotowe pliki typu .csv czy .json zawierające jeszcze niedawno niemożliwe do zdobycia, wartościowe informacje wręcz zalewają sieć Internet. Nie inaczej przedstawia się sprawa z dostępnością do oprogramowania. Darmowe alternatywy jeśli nie zastępują ich własnościowych odpowiedników, to przynajmniej pokrywają coraz to nowe przypadki użycia w sposób szczególnie istotny dla przeciętnych użytkowników, których ze względów finansowych nie stać na płatne licencje. Wolne inicjatywy takie jak OpenStreetMap czy Folium dokładają swoją cegiełkę do ekosystemu narzędzi geoprzestrzennych, które w rezultacie przekładają się na głębsze zrozumienie realiów otaczającego nas świata.

2.3. Kluczowe aspekty teoretyczne systemów GIS

2.3.1. Lokalizacja

Podstawowym elementem wszelkiego rodzaju Systemów Informacji Geograficznej jest określenie położenia danego obiektu na mapie. Najbardziej powszechnym sposobem umiejscawiania jest wykorzystanie szerokości i długości geograficznej. Jeśli rozważymy bardzo popularne, tzw. cylindryczne rzutowanie Ziemi (O rzutowaniach więcej w podrozdziale 2.3.2), to współrzędne geograficzne możemy potraktować jako osie dwuwymiarowego wykresu mapy. Wtedy szerokość geograficzną odczytujemy z osi y, a długość z osi x (rys. 2.1).



Rys. 2.1: Przykład cylindrycznej projekcji Ziemi [21]

Dodatkowym, wartym wspomnienia, aspektem geolokalizacji jest również możliwość wystąpienia trzeciej osi - osi z. Posługuje się nią w celu określenia elewacji danego obiektu nad powierzchnią planety. Zagadnienia związane z elewacją znajdują się poza zakresem tej pracy.

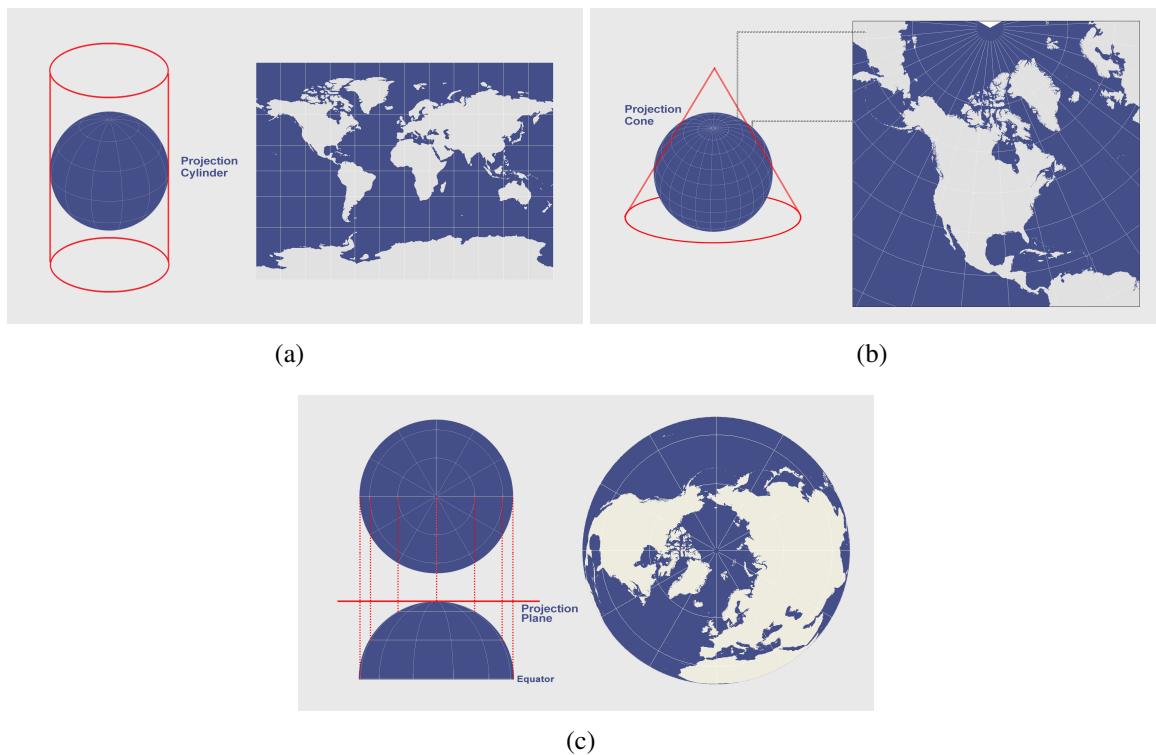
2.3.2. Odwzorowania geograficzne

Odwzorowaniem geograficznym (rzutowaniem) nazywamy utworzenie dwuwymiarowej mapy Ziemi z trójwymiarowego kształtu planety przy użyciu matematycznej transformacji. Wszystkie modele rzutowania mają jedną wspólną cechę - są mniejszym lub większym uproszczeniem i zakrzywieniem rzeczywistej geometrii kuli ziemskiej. Niektóre z nich przedstawiają pewne obszary Ziemi z bardzo dużą dokładnością, jednocześnie całkowicie przekładając skalę, rozmiar i odległości innych. Inne typy rzutowań skupiają się na jak najwierniejszym oddaniu powierzchni obszarów przy jednocześnie całkowitym zniekształceniu ich kształtów, a jeszcze inne na zupełnie odwrotnym rezultacie [33]. Do najważniejszych typów odwzorowań należą:

- a) rzutowanie cylindryczne (walcowe) - zaprezentowane już w rozdziale 2.3.1 na przykładzie rzutu poprzecznego Merkatora. Uzyskiwane poprzez "rozwiniecie" ściany bocznej walca, na który rzutowano kulę ziemską. Cechuje je pogarszająca się wierność skali mapy wraz z oddalaniem się od równika w kierunku biegunów (rys. 2.2(a));
- b) rzutowanie stożkowe - uzyskiwane poprzez rzutowanie kuli ziemskiej na powierzchnię boczną stożka. Największą dokładność uzyskuje się na wysokości styku kuli z ścianą boczną stożka (rys. 2.2(b));

- c) rzutowanie azymutalne - polega na bezpośrednim odwzorowaniu powierzchni Ziemi na płaski, okrągły dysk. Tego typu rzutowanie jest skoncentrowane wokół wybranego punktu na kuli. Najlepszą dokładność uzyskuje się w okolicy wybranego punktu centralnego, a pogarsza się wraz z oddalaniem od niego (rys. 2.2(c)).

W zakres projektu inżynierskiego weszła wyłącznie praca z rzutowaniem cylindrycznym.



Rys. 2.2: Porównanie rzutowań: (a) cylindrycznego, (b) stożkowego, (c) azymutalnego [22]

2.3.3. Dane wektorowe

Powszechnie uważa się, że dane wektorowe są najbardziej wydajnym sposobem zapisywania informacji geoprzestrzennych. W porównaniu do danych rastrowych są znacznie bardziej efektywne pod względem złożoności obliczeniowej i pamięciowej. Dane wektorowe tworzone są przez kształty geometryczne - punkty, linie oraz wielokąty. Punkty, jako najbardziej prymitywna struktura, tworzą linie oraz wielokąty, które następnie pod postacią zbioru tychże punktów przechowywane są w plikach [32]. Do najbardziej powszechnych typów danych wektorowych zaliczamy:

- GeoJSON** - stosunkowo nowy, tekstowy standard przechowywania danych geoprzestrzennych wywodzący się od popularnego formatu JSON, służącego do łatwiej dystrybucji danych [32];
- Języki znaczników - obecne zazwyczaj pod postacią formatów **XML**, które służą do przedstawiania danych w zorganizowany, ustrukturyzowany sposób. Ich najważniejszą cechą jest rozszerzalność, która umożliwia dostosowanie struktury przechowywania danych do konkretnego problemu. Ta kategoria danych wektorowych zostanie omówiona na przykładzie formatu stworzonego na potrzeby API **OpenStreetMap**;
- Shapefile** - dane w tym formacie składają się z co najmniej trzech plików o rozszerzeniach .shp, .shx, .dbf (może wystąpić jeszcze dwanaście dodatkowych plików rozszerzających funkcjonalność formatu Shapefile). Plik .shp zawiera całą geometrię danych, .shx pełni

rolę pliku indeksującego zawartość .shp w celu przyśpieszenia dostępu, .dbf to bazodanowy plik zawierający atrybuty geometrii z .shp [32]. Format nie będzie szerzej omawiany, ponieważ wykracza poza zakres narzędzi wykorzystanych w projekcie.

GeoJSON

Standard ten jest całkowicie kompatybilny z formatem JSON, którego stanowi de facto rozszerzenie. Co za tym idzie, znaczco ułatwia to interakcję ze skryptami napisanymi w języku Python, ze względu na jego podobieństwo do struktur słownika oraz listy. Największymi zaletami kodu w standardzie GeoJSON są niewątpliwie jego czytelność, szerokie zastosowanie w branży IT i bardzo dobrze rozwinięte wsparcie w różnego rodzaju oprogramowaniu.

Listing 2.1: Przykładowy kod GeoJSON

```

1 {
2   "type": "FeatureCollection",
3   "features": [
4     {
5       "type": "Feature",
6       "geometry": {
7         "type": "Point",
8         "coordinates": [102.0, 0.5]
9       },
10      "properties": {
11        "prop0": "value0"
12      }
13    },
14    {
15      "type": "Feature",
16      "geometry": {
17        "type": "LineString",
18        "coordinates": [
19          [102.0, 0.0], [103.0, 1.0], [104.0, 0.0], [105.0, 1.0]
20        ]
21      },
22      "properties": {
23        "prop0": "value0",
24        "prop1": 0.0
25      }
26    },
27    {
28      "type": "Feature",
29      "geometry": {
30        "type": "Polygon",
31        "coordinates": [
32          [
33            [100.0, 0.0], [101.0, 0.0], [101.0, 1.0],
34            [100.0, 1.0], [100.0, 0.0]
35          ]
36        ]
37      },
38      "properties": {
39        "prop0": "value0",
40        "prop1": { "this": "that" }
41      }
42    }
43  ]
44 }
```

Powyższy przykład (listing 2.1) pokazuje, że wszystkie elementy muszą zawierać się wewnątrz obiektu *FeatureCollection*. Dla każdego elementu w polu *geometry* określamy typ danych do przechowania. W sumie dostępne jest siedem różnych opcji, które dzielą się na dwie kategorie - typy prymitywne przedstawione na listingu (*Point*, *LineString* i *Polygon*) oraz typy wieloczęściowe (*MultiPoint* - zbiór punktów, *MultiLine* - zbiór linii, *MultiPolygon* - zbiór wielokątów, *GeometryCollection* - zbiór prymitywów różnego rodzaju) [23].

Ze względu na swoją kompatkowość, przyjazną składnię i szerokie wsparcie w największych językach programowania, GeoJSON stał się kluczowym elementem komunikacji dla geoprzestrzennych REST API. Uważa się, że format ten oferuje obecnie optymalny kompromis pomiędzy czytelnością kodu, wydajnością i programistyczną użytecznością [32].

OSM XML

Format ten stworzono w celu zapewnienia łatwej dystrybucji usługi kartograficznej OpenStreetMap. Wszystkie atrybuty zapisywane są w postaci znaczników. Posiada on cztery klasy danych:

- *node* - informacja o pojedynczych punktach,
- *way* - sekwencja punktów tworząca linię,
- *area* - "domknięta" klasa *ways* reprezentująca wielokąt,
- *relation* - zbiór wyżej wymienionych elementów.

Dodatkowo istnieje jeszcze jedno słowo kluczowe *tag*, którego można używać w sposób nieograniczony do dostarczenia dodatkowych informacji o obiekcie danej klasy [33]. Przykład struktury pliku przedstawia listing 2.2

Listing 2.2: Przykładowy kod OSM XML

```

1 <osm>
2     <node id="603279517" lat="-38.1456457" lon="176.2441646" . . . />
3     <node id="603279518" lat="-38.1456583" lon="176.2406726" . . . />
4     <node id="603279519" lat="-38.1456540" lon="176.2380553" . . . />
5     ...
6     <way id="47390936" . . . >
7         <nd ref="603279517"/>
8         <nd ref="603279518"/>
9         <nd ref="603279519"/>
10        <tag k="highway" v="residential"/>
11        <tag k="name" v="York Street"/>
12    </way>
13    ...
14    <relation id="126207" . . . >
15        <member type="way" ref="22930719" role="" />
16        <member type="way" ref="23963573" role="" />
17        <member type="way" ref="28562757" role="" />
18        <member type="way" ref="23963609" role="" />
19        <member type="way" ref="47475844" role="" />
20        <tag k="name" v="State Highway 30A"/>
21        <tag k="ref" v="30A"/>
22        <tag k="route" v="road"/>
23        <tag k="type" v="route"/>
24    </relation>
25 </osm>
```

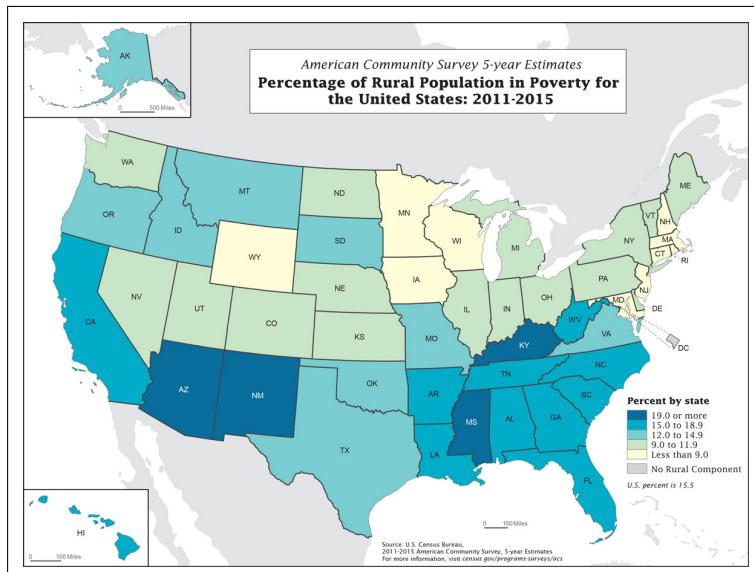
2.3.4. Wybrane typy wizualizacji kartograficznych

Mapy towarzyszą gatunkowi ludzkiemu już od czasów starożytnej Grecji, zaczynając od około VII w. p.n.e [12], aczkolwiek dopiero w erze cyfrowej przeistoczyły one się z wyłącznie statycznych, drukowanych kawałków papieru do cyfrowych, interaktywnych oraz elastycznych narzędzi oceny rzeczywistości. Możliwość spojrzenia na "zmapowane" dane zarówno podtrzymała i wzmacniła wcześniej posiadaną wiedzę, ale również dostarczyła dodatkowego, cennego, wcześniej nieznanego kontekstu [10]. Oczywiście aby zwizualizowane dane sprostały swojemu przeznaczeniu powinny być przedstawione w odpowiedni, uporządkowany i najbardziej efektywny dla posiadanej przez siebie typu informacji, sposób.

Kartogram

Kartogram jest jednym z najbardziej popularnych typów wizualizacji danych tematycznych. Jego istotą jest powiązanie pewnych informacji z obszarem geograficznym, np. państwu, regionem administracyjnym czy miastem. Kartogramy bardzo często wykorzystuje się do kartograficznych reprezentacji danych statystycznych. Używa się do tego celu, zazwyczaj, palet kolorów, które poprzez pewną progresję, np. z jaśniejszego odcienia do ciemniejszego, reprezentują zmianę wartości określonego wskaźnika.

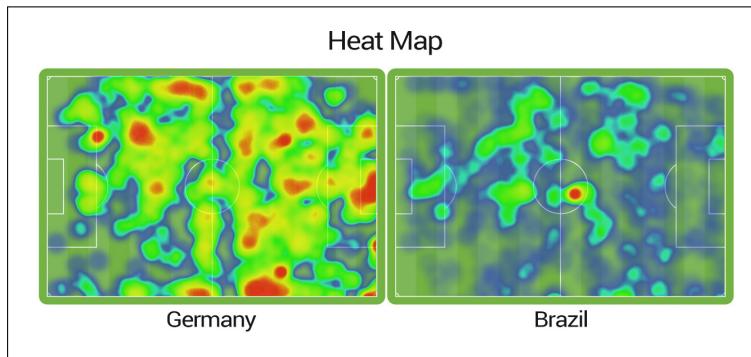
Największą wadą tego typu wizualizacji jest brak możliwości dokładnego porównania wartości poszczególnych regionów mapy, ponieważ ich przedstawienie za pomocą kolorów jest uproszczonym modelem rzeczywistości. Z tego względu dane prezentowane na kartogramie są wysoce narażone na manipulację i niezbędnym elementem każdego kartogramu jest co najmniej legenda informująca jakim kryterium przedziałowi wartości odpowiada dana barwa na mapie (rys. 2.3).



Rys. 2.3: Przykład kartogramu [14]

Mapa termiczna

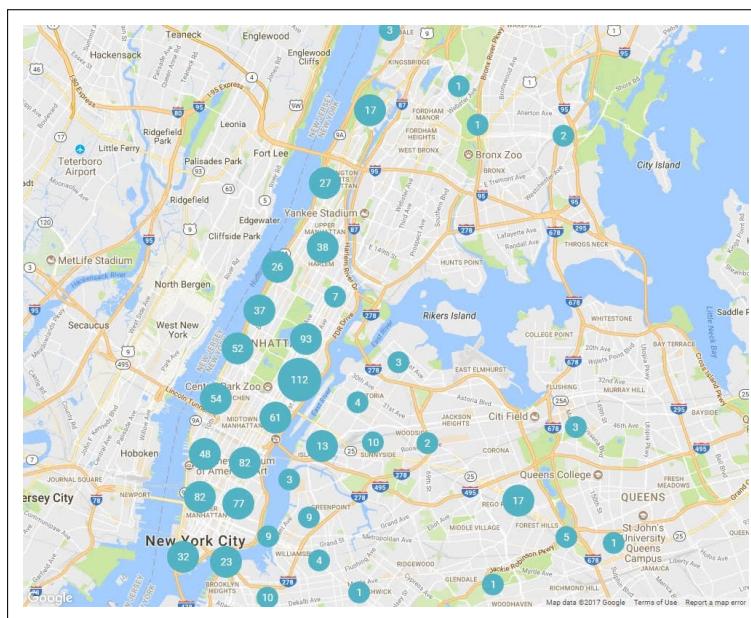
W przypadku mapy termicznej również mamy do czynienia z mapowaniem informacji za pomocą kolorów, jednakże nie jest ono oparte o pozycję na skali wartości, a o zagęszczenie danych znajdujących się w sąsiedztwie względem siebie. Mapy termiczne są szczególnie przydatne w odkrywaniu nieznanych wcześniej trendów, które są powiązane z pewnymi obszarami geograficznymi (rys. 2.4).



Rys. 2.4: Mapa termiczna na przykładzie pamiętnego meczu Brazylia - Niemcy, półfinał mundialu 2014. Bardziej zabawna niż rzetelna (aczkolwiek dobrze obrazująca istotę map termicznych) wizualizacja czasu posiadania piłki w danych obszarach boiska przez obie drużyny [29]

Klaster znaczników

Znaczniki same w sobie są powszechnie używanym narzędziem do wyraźnego wskazywania położenia danego obiektu/informacji na mapie. Klastry znajdują zastosowanie w chwili, gdy liczba znaczników staje się bardzo duża, a ich grafiki zaczynają przesłaniać znaczną część mapy, pogarszając przejrzystość wizualizacji. Są one niczym innym jak zgrupowaniem znaczników na pewnym obszarze geograficznym i informacją o ich ilości na tymże obszarze (rys. 2.5).



Rys. 2.5: Przykład klastra znaczników na mapie Nowego Jorku [28]

Rozdział 3

Przegląd i analiza źródeł danych

W tym rozdziale omówiona została faza wstępna procesu tworzenia wizualizacji, jaką jest zebranie interesujących i rzetelnych danych, spełniających podstawowe wymogi geoprzestrzenne w zakresie ich formatu oraz zawartości. W ramach projektu udało się zgromadzić oraz z powodzeniem zwizualizować (więcej o wizualizacjach w rozdziale 4) cztery zbiory danych. Najważniejszym, subiektywnym kryterium w doborze źródeł informacji była chęć przedstawienia wielu różnych metod tworzenia wizualizacji. Mowa tutaj o zróżnicowaniu ze względu na kategorię wizualizacji, jej interaktywność, dynamiczność lub statyczność oraz przedział czasowy, jakiego dotyczy.

3.1. Geometria granic

Dla każdej wizualizacji, która opiera się na przedstawieniu danych powiązanych z pewnym regionem geograficznym, niezbędny jest plik zawierający geometrię tychże regionów (patrz rozdział 2.3.3). W przypadku omawianego projektu inżynierskiego, dwie z czterech wizualizacji opierają się o prezentację danych z podziałem na kraje, których dotyczą. Z tego względu jeszcze przed podjęciem jakichkolwiek działań związanych z manipulacją danych docelowych, należało zadbać o zdobycie i przygotowanie źródła zawierającego kształty wszystkich państw. Plik o formacie GeoJSON spełniający te założenia został pobrany ze strony <https://www.naturalearthdata.com>. Korzystanie z pliku opiera się o licencję *Open Data Commons* [26].

Listing 3.1: Fragment pliku GeoJSON zawierającego geometrię granic państw świata

```
1 {  
2     "type": "FeatureCollection",  
3     "features": [  
4         {  
5             "type": "Feature",  
6             "properties": {  
7                 "ADMIN": "Aruba",  
8                 "ISO_A3": "ABW",  
9                 "ISO_A2": "AW"  
10            },  
11            "geometry": {  
12                "type": "MultiPolygon",  
13                "coordinates": [  
14                    [  
15                        [  
16                            [  
17                                -69.996937628999916,
```

```

18                     12.577582098000036
19                 ],
20                 [
21                     -69.936390753999945,
22                     12.531724351000051
23                 ],
24                 [
25                     -69.924672003999945,
26                     12.519232489000046
27                 ],
28                 [
29                     -69.915760870999918,
30                     12.497015692000076
31                 ],
32                 [
33                     -69.880197719999842,
34                     12.453558661000045
35                 ],
36                 ...
37             ]
38             ...
39         ]
40         ...
41     ]
42 }
43 ...
44 ...
45 ]
46 }

```

Kod załączony w listingu 3.1 pokazuje schemat pliku geometrii granic, na przykładzie państwa Aruba. Kluczowymi wartościami są tutaj właściwości "ADMIN" (pełna nazwa państwa) oraz "ISO_A3". ISO_A3, a właściwie kod standardu ISO 3166-1 alfa 3 [20] jest trzynakową notacją pełniącą rolę unikalnego identyfikatora danego regionu administracyjnego (w tym przykładzie państwa).

Jednym istotnym problemem z informacjami zawartymi w pobranym pliku GeoJSON jest fakt, że wąskie grono państw nie posiada wprowadzonych unikalnych identyfikatorów ISO_A3 (np. Kosowo, Tajwan, przeróżne regiony sporne). W takim wypadku dobrą praktyką jest albo manualne wypełnienie luk, lub też nadpisanie ich w procesie wewnętrznego przetwarzania danych docelowych kodami zawartymi w tychże danych (które czasami mieszczą w sobie informację o kodzie ISO alfa 3 danego regionu).

3.2. Produktu krajowy brutto per capita wyrażony w USD (lata 1960-2019)

Zbiór danych zaczerpnięto z portalu data.worldbank [13], na którym jest on dostępny pod postacią plików o formatach .csv, .xml oraz .xlsx. Ze względu na bardzo dobre wsparcie wczytywania dokumentów .csv przez moduł *Pandas* wybrany został właśnie wspomniany format. Dostęp do danych jest oparty o licencję *Creative Commons Attribution 4.0*, która, nie licząc pewnych wyjątków, umożliwia właściwie dowolne kopiowanie, modyfikowanie i dystrybucję danych, również w celach komercyjnych [25]. Po pobraniu pliku rozpoczęto właściwą procedurę analizy struktury zbioru (rys. 3.1).

Country Name	Country Code	Indicator Name	Indicator Code	1960	1961	1962	1963
Aruba	ABW	GDP per capita (current US\$)	NY.GDP.PCAP.CD				
Afghanistan	AFG	GDP per capita (current US\$)	NY.GDP.PCAP.CD	59.7731938409853	59.8608738790779	58.4580149495439	78.7063875407802
Angola	AGO	GDP per capita (current US\$)	NY.GDP.PCAP.CD				
Albania	ALB	GDP per capita (current US\$)	NY.GDP.PCAP.CD				
Andorra	AND	GDP per capita (current US\$)	NY.GDP.PCAP.CD				
Arab World	ARB	GDP per capita (current US\$)	NY.GDP.PCAP.CD				
United Arab Emirates	ARE	GDP per capita (current US\$)	NY.GDP.PCAP.CD				
Argentina	ARG	GDP per capita (current US\$)	NY.GDP.PCAP.CD			1155.89017025099	850.30473690206
Armenia	ARM	GDP per capita (current US\$)	NY.GDP.PCAP.CD				
American Samoa	ASM	GDP per capita (current US\$)	NY.GDP.PCAP.CD				
Antigua and Barbuda	ATG	GDP per capita (current US\$)	NY.GDP.PCAP.CD				
Australia	AUS	GDP per capita (current US\$)	NY.GDP.PCAP.CD	1807.78571021206	1874.73210577126	1851.84185074588	1964.1504696359
Austria	AUT	GDP per capita (current US\$)	NY.GDP.PCAP.CD	935.460426850415	1031.8150043291	1087.8342434189	1167.00053244585
Azerbaijan	AZE	GDP per capita (current US\$)	NY.GDP.PCAP.CD				
Burundi	BDI	GDP per capita (current US\$)	NY.GDP.PCAP.CD	70.0517346382971	71.1671882088235	73.4353055613742	78.5143294134077
Belgium	BEL	GDP per capita (current US\$)	NY.GDP.PCAP.CD	1273.69165910289	1350.19767333123	1438.5232330684	1535.02372901043
Benin	BEN	GDP per capita (current US\$)	NY.GDP.PCAP.CD	93.0225089907107	95.5721547147448	94.4645349843821	99.8591138855553
Burkina Faso	BFA	GDP per capita (current US\$)	NY.GDP.PCAP.CD	68.424748569325	71.55818092437	76.5206113987061	78.372071942432
Bangladesh	BGD	GDP per capita (current US\$)	NY.GDP.PCAP.CD	89.0352412832723	97.5952739108149	100.12211579079	101.901414464052
Bulgaria	BGR	GDP per capita (current US\$)	NY.GDP.PCAP.CD				
Bahrain	BHR	GDP per capita (current US\$)	NY.GDP.PCAP.CD				
Bahamas, The	BHS	GDP per capita (current US\$)	NY.GDP.PCAP.CD	1550.23939204838	1651.28898476982	1752.85448101663	1867.02501228387
Bosnia and Herzegovina	BIH	GDP per capita (current US\$)	NY.GDP.PCAP.CD				
Belarus	BLR	GDP per capita (current US\$)	NY.GDP.PCAP.CD				
Belize	BLZ	GDP per capita (current US\$)	NY.GDP.PCAP.CD	304.917107253962	316.403606143266	327.126867468401	336.94146611457

Rys. 3.1: Zrzut ekranu przedstawiający fragment pliku zawierającego dane o PKB per capita

Spośród wszystkich kolumn w analizowanej strukturze danych wyszczególnić można w zasadzie ich trzy typy, które były potrzebne do wizualizacji. Pierwszym, jest pełna nazwa danego państwa. Plik .csv zawiera także rekordy dotyczące uogólnionych regionów takich jak Centralna Europa i państwa nadbałtyckie czy państwa arabskie. Dane te nie są w tym przypadku przydatne i zostały usunięte w fazie wstępnego przetwarzania (rozdział 4.2).

Drugim istotnym typem informacji jest kolumna dotycząca unikalnego identyfikatora danego państwa. W przypadku tego zbioru danych rolę tę pełni wspomniany w rozdziale 3.1 kod ISO_A3.

Ostatnią pożądaną informacją, a w zasadzie ich zbiorem, są kolumny odpowiadające kolejnym latom od roku 1960 do 2019. Łatwo zauważyc, że w tej kategorii istnieje sporo luk i należało to odzwierciedlić na wizualizacji poprzez odpowiednie oznaczenia obszarów geograficznych, dla których występują.

3.3. Pandemia wirusa COVID-19 na świecie (stan obecny)

Źródłem wykorzystanych danych epidemicznych jest repozytorium Centrum Systemów Naukowych i Inżynieryjnych Uniwersytetu im. Johna Hopkina w Baltimore [17], [31]. Dane dot. rozwoju epidemii pojawiają się tam w cyklu dobowym, pod postacią plików o szablonie "[dzień]-[miesiąc]-[rok].csv". Twórcy repozytorium określają godziny, w których następuje aktualizacja danych, jako przedział pomiędzy 4:45, a 5:15 czasu GMT. Komunikacja ze stroną została dokładniej opisana w rozdziale 4.3. Dostęp do danych jest oparty o licencję *Creative Commons Attribution 4.0* [25]. Po pobraniu pliku rozpoczęto właściwą procedurę analizy struktury zbioru (rys. 3.2).

3.4. Przestępcość w San Francisco (ostatnie siedem dni)

FIPS	Adm	Province_State	Country_Region	Last_Update	Lat	Long_	Confirmed	Deaths	Recovered	Active	Combined_Key	Incident_Rate	Case_Fatality_Ratio
		Afghanistan	Afghanistan	02.12.2020 05:27 33.93911	67.709953	46717	1797	36907	8013	Afghanistan	120.00768283446756	3.846565490078558	
		Albania	Albania	02.12.2020 05:27 41.1533	20.1683	39014	822	19384	7015	Albania	135.688330267562	2.10659717024656	
		Algeria	Algeria	02.12.2020 05:27 28.0339	1.6596	84152	2447	54405	27300	Algeria	191.90421536837792	2.907833442437495	
		Andorra	Andorra	02.12.2020 05:27 42.5063	1.5218	6790	76	5940	774	Andorra	8787.937617291143	1.119293070559647	
		Angola	Angola	02.12.2020 05:27 11.2027	17.8739	15251	350	7932	6969	Angola	46.40319977917785	2.294931479902957	
		Antigua and Barbuda	Antigua and Barbuda	02.12.2020 05:27 17.0608	-61.7964	142	4	130	8	Antigua and Barbuda	145.00449309696918	2.816901408450704	
		Argentina	Argentina	02.12.2020 05:27 38.4161	-63.8167	142570	38928	1263521	130391	Argentina	3169.6987973013497	2.7173541258018807	
		Armenia	Armenia	02.12.2020 05:27 40.6691	45.0382	135967	2193	110365	23409	Armenia	4588.466520025081	1.6128913633455177	
		Australian Capital Territory	Australia	02.12.2020 05:27 34.9285	149.0124	117	3	112	2	Australian Capital Territory, Australia	27.330063069376315	2.5641025641025643	
		New South Wales	Australia	02.12.2020 05:27 -33.8688	151.2093	4588	53	3183	1552	New South Wales, Australia	56.51638334565163	1.1551874455100262	
		Northern Territory	Australia	02.12.2020 05:27 -12.4634	130.8456	53	0	46	7	Northern Territory, Australia	21.579804562065087	0.0	
		Queensland	Australia	02.12.2020 05:27 27.4698	153.0251	1205	6	1186	13	Queensland, Australia	23.555859642263712	0.4972953112033195	
		South Australia	Australia	02.12.2020 05:27 35.4735	138.6007	562	4	548	10	South Australia, Australia	31.99544548818672	0.7117437722419929	
		Tasmania	Australia	02.12.2020 05:27 -42.8821	147.3372	230	13	217	0	Tasmania, Australia	42.59051353874883	5.6521739130434785	
		Victoria	Australia	02.12.2020 05:27 -37.8133	144.9631	20345	820	19525	0	Victoria, Australia	306.8673735652121	4.030474318014254	
		Western Australia	Australia	02.12.2020 05:27 -31.9504	115.8605	823	9	789	25	Western Australia, Australia	31.285638257431764	1.0935601458080195	
		Austria	Austria	02.12.2020 05:27 47.1562	14.5501	285489	3325	227497	54667	Austria	3169.845887368982	1.164668340986681	
		Azerbaijan	Azerbaijan	02.12.2020 05:27 40.1431	47.5769	125602	1433	76897	47272	Azerbaijan	1238.7792892419748	1.140903395955478	
		Bahamas	Bahamas	02.12.2020 05:27 25.05885	-78.035889	7543	163	5934	1446	Bahamas	1918.1280006509887	2.1609439215166377	
		Bahrain	Bahrain	02.12.2020 05:27 36.0375	50.55	87137	341	85357	1439	Bahrain	5120.937385951787	0.39133777844094605	
		Bangladesh	Bangladesh	02.12.2020 05:27 23.6858	90.3563	67225	6675	383224	77326	Bangladesh	283.7007410489843	1.4286478677299806	
		Barbados	Barbados	02.12.2020 05:27 13.1936	-59.5432	278	7	255	16	Barbados	96.739058568883	2.5179856115107913	
		Belarus	Belarus	02.12.2020 05:27 53.7098	27.9534	138219	1166	115987	21466	Belarus	1461.7400212142227	0.8435887974880444	
		Antwerp	Belgium	02.12.2020 05:27 51.2195	4.4024	68099	0	68099	68099	Antwerp, Belgium	3665.205227595902	0.0	
		Brussels	Belgium	02.12.2020 05:27 50.8503	4.3517	76573	0	76573	76573	Brussels, Belgium	6335.981703573397	0.0	
		East Flanders	Belgium	02.12.2020 05:27 51.0362	3.7373	54243	0	54243	54243	East Flanders, Belgium	3580.24479439356	0.0	
		Flemish Brabant	Belgium	02.12.2020 05:27 50.9167	4.5833	44133	0	44133	44133	Flemish Brabant, Belgium	3850.459135621319	0.0	
		Hainaut	Belgium	02.12.2020 05:27 50.5357	4.0521	93621	0	93621	93621	Hainaut, Belgium	6964.5993538301045	0.0	
		Liège	Belgium	02.12.2020 05:27 50.4496	5.8492	91816	0	91816	91816	Liège, Belgium	8294.188214548976	0.0	
		Limburg	Belgium	02.12.2020 05:27 50.7939	5.3420000C	24501	0	24501	24501	Limburg, Belgium	2803.1641282858614	0.0	
		Luxembourg	Belgium	02.12.2020 05:27 50.0547	5.4677	16344	0	16344	16344	Luxembourg, Belgium	5742.20302786838	0.0	
		Namur	Belgium	02.12.2020 05:27 50.3311	4.8221	32434	0	32434	32434	Namur, Belgium	6561.270419258585	0.0	
		Unknown	Belgium	02.12.2020 05:27		9561	16786	0	-7225	Unknown, Belgium	175.5674092668131		

Rys. 3.2: Zrzut ekranu przedstawiający fragment pliku zawierającego dane o epidemii wirusa COVID-19

Pierwsze dwie kolumny to dane obowiązujące wyłącznie wewnątrz Stanów Zjednoczonych i zostały one pominięte. Niektóre z państw rozdzielono na poszczególne regiony wewnętrzne, stąd obecność kolumny "Province_State". Tego typu strukturę należało zagregować w celu uzyskania pojedynczego rekordu dla danego państwa. Ponieważ wizualizacja pandemii korzysta z najnowszego dostępnego zbioru danych, konieczna jest jej cykliczna aktualizacja (więcej w rozdziale 4.2) i informacja zawarta w kolumnie "Last_Update" przydaje się do informowania użytkownika o tym jak aktualne są przedstawiane mu informacji. Długość i szerokość geograficzna z "Long_" oraz "Lat", a także pole "Combined_Key" zostały pominięte ze względu na swoją nieużyteczność dla wizualizacji.

Kolumny, które pozostały, dotyczą konkretnych danych statystycznych z poszczególnych regionów. Spośród sześciu z nieomówionych dotąd kolumn wykorzystanych zostało pięć. Atrybut "Recovered" jest o tyle problematyczny, że dla wielu państw nie jest w ogóle aktualizowany i to zadecydowało o jego porzuceniu. "Confirmed", "Deaths" oraz "Active" ozaczają kolejno liczbę odnotowanych przypadków zachorowań, liczbę zgonów oraz liczbę aktywnych przypadków. "Incident_Rate" to współczynnik zachorowań na 100 000 obywateli. "Case_Fatality_Ratio" to procentowa wartość z podzielenia liczby odnotowanych zgonów przez liczbę odnotowanych przypadków [17].

Warto zaznaczyć, że agregacja omówionych danych statystycznych przebiegała inaczej w zależności od parametru. W przypadku kolumn zliczających poszczególne zdarzenia zastosowana została suma, a w kolumnach dotyczących współczynników należało wyznaczyć średnią.

3.4. Przestępcość w San Francisco (ostatnie siedem dni)

Kolejne z wykorzystanych źródeł zostało zaczerpnięte ze strony data.sfgov.org [19], która udostępnia dane związane z hrabstwem San Francisco pod jurysdykcją lokalnych władz. Konkretny zbiór wykorzystany do stworzenia wizualizacji aktywności przestępcozej na terenie San Francisco pochodzi od miejskiego departamentu policji. Aktualizacja danych na stronie następuje codziennie o godzinie 10:00 czasu PT i dokładna komunikacja z nią zostanie omówiona w rozdziale 4.3. Dostęp do danych opiera się o licencję Open Data Commons [26]. Strona internetowa zapewnia użytkownikowi elegancki sposób na przeglądanie struktury interesujących go zbiorów danych, dlatego w tym przypadku wstępna analiza nie wymagała uprzedniego pobrania pliku [19] (rys. 3.3).

3.4. Przestępcość w San Francisco (ostatnie siedem dni)

Column Name	Description	Type
Incident Datetime	The date and time when the incident occurred	Date & Time 
Incident Date	The date the incident occurred	Date & Time 
Incident Time	The time the incident occurred	Plain Text 
Incident Year	The year the incident occurred, provided as a convenience ...	Plain Text 
Incident Day of Week	The day of week the incident occurred	Plain Text 
Report Datetime	Distinct from Incident Datetime, Report Datetime is when t...	Date & Time 
Row ID	A unique identifier for each row of data in the dataset	Plain Text 
Incident ID	This is the system generated identifier for incident reports. ...	Plain Text 
Incident Number	The number issued on the report, sometimes interchangea...	Plain Text 
CAD Number	The Computer Aided Dispatch (CAD) is the system used by ...	Plain Text 
Report Type Code	A system code for report types, these have corresponding ...	Plain Text 
Report Type Description	The description of the report type, can be one of: Initial; Ini...	Plain Text 
Filed Online	Non- emergency police reports can be filed online by mem...	Checkbox 
Incident Code	Incident Codes are the system codes to describe a type of i...	Plain Text 
Incident Category	A category mapped on to the Incident Code used in statisti...	Plain Text 
Incident Subcategory	A subcategory mapped to the Incident Code that is used fo...	Plain Text 
Incident Description	The description of the incident that corresponds with the i...	Plain Text 
Resolution	The resolution of the incident at the time of the report. Can...	Plain Text 
Intersection	The 2 or more street names that intersect closest to the ori...	Plain Text 
CNN	The unique identifier of the intersection for reference back ...	Plain Text 
Police District	The Police District where the incident occurred. District bo...	Plain Text 
Analysis Neighborhood	This field is used to identify the neighborhood where each i...	Plain Text 
Supervisor District	There are 11 members elected to the Board of Supervisors ...	Plain Text 
Latitude	The latitude coordinate in WGS84, spatial reference is EPS...	Number 
Longitude	The longitude coordinate in WGS84, spatial reference is EP...	Number 
point	The point geometry used for mapping features in the open...	Location 

Rys. 3.3: Struktura zbioru danych dot. przestępcości w San Francisco, [19] [dostęp dnia 03.12.2020]

Jak widać na powyższej grafice, departament policji w San Francisco zdaje bardzo precyzyjną relację z zanotowanych wykroczeń, co niewątpliwie świadczy o niebagatelnej skrupulatności tamtejszych urzędników, aczkolwiek większość z dostarczonych informacji nie znalazła zastosowania w wizualizacji opartej o model klastra znaczników. Użyto tylko takich kategorie danych jak długość i szerokość geograficzna miejsca, w którym doszło do wykroczenia, jego data i czas, a także krótki opis tła zdarzenia, który wyświetli się w momencie wybrania danego znacznika na mapie.

Szczególną uwagę należało zwrócić na kwestię obecności informacji o miejscu przestępstwa. W przypadku jej braku nie ma możliwości włączenia takowego zdarzenia do wizualizacji i rekord, którego dotyczy, należało usunąć w procesie wstępnego przetwarzania. Ponieważ w założeniach opisywanego podprojektu jest praca z danymi pochodząymi z ostatniego tygodnia, koniecznym było zastosowanie odpowiedniej filtracji zdarzeń na podstawie daty ich wystąpienia.

3.5. Wypadki drogowe w Wielkiej Brytanii (rok 2015)

Źródłem dla ostatniego już podprojektu jest zbiór danych w formacie .csv zamieszczony na stronie rządowej Wielkiej Brytanii data.gov.uk [18]. Plik zawiera informacje o wypadkach, które wydarzyły się w latach 2011-2015, ale ze względu na chęć minimalizacji obciążenia pamięciowego użyte zostały wyłącznie dane z roku 2015. Dostęp do danych reguluje licencja Open Government License [27]. Po pobraniu pliku rozpoczęto właściwą procedurę analizy struktury zbioru (rys. 3.4).

Accident_Index	Longitude	Latitude	Accident_Severity	Date
2015160A04651	-0.08068600000000001	53.55978399999999	2	31.12.2015
2015200038986	-1.9417520000000001	52.513103	3	31.12.2015
2015320004197	0.335502	53.23109599999999	3	31.12.2015
2015350220215	-0.2230599999999998	52.605773	3	31.12.2015
2015320005836	-0.0310149999999997	52.97586800000001	3	31.12.2015
201514C142615	-1.397265	53.431194	3	31.12.2015
2015530039029	-2.075733	51.900461	3	31.12.2015
2015200039279	-1.8793540000000002	52.468295	3	31.12.2015
2015350220015	0.159602	52.65399100000001	2	31.12.2015
201501MM79238	-0.055798	51.499034	3	31.12.2015
2015200039247	-2.1108700000000002	52.502088	3	31.12.2015
201501LX59021	-0.1355130000000002	51.462739	2	31.12.2015
2015160C07841	-0.20274	54.09688199999994	3	31.12.2015
2015051503513	-2.98457	53.432033	3	31.12.2015
201501MM71133	-0.101172	51.49582	3	31.12.2015
201534WN34035	-0.867764	52.25105900000001	3	31.12.2015
2015160A04661	-0.1245609999999999	53.551301	3	31.12.2015
2015320144247	-0.028411000000000002	52.97450300000001	3	31.12.2015
2015200039914	-1.8462869999999998	52.512002	2	31.12.2015
201563DP42915	-3.4420830000000002	52.214183	3	31.12.2015
2015950005983	-3.624472	55.98213399999995	3	31.12.2015
2015471507743	-0.1320849999999998	50.830268	3	31.12.2015

Rys. 3.4: Zrzut ekranu przedstawiający fragment pliku zawierającego dane o wypadkach drogowych w Wielkiej Brytanii w roku 2015

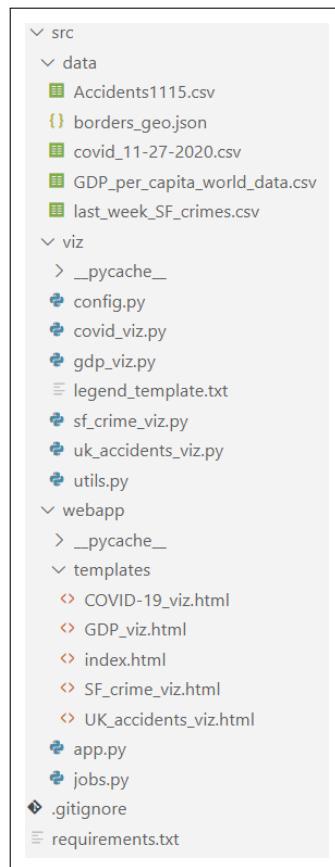
Powyższy zbiór wykorzystano do stworzenia mapy termicznej, a z racji jej specyfiki nie ma mowy o interakcji z pojedynczymi zdarzeniami ją tworzącymi. Z tego powodu wykorzystane zostały tylko trzy kolumny - szerokość i długość geograficzna oraz data zdarzenia. Wszystkie z wymienionych kolumn są niezbędne do poprawnego wygenerowania wybranego typu wizualizacji, dlatego należało usunąć każdy rekord, w którym chociaż jedna z tych informacji nie została wprowadzona.

Rozdział 4

Implementacja projektu

Po dokonaniu doboru i wstępnej analizy danych poddawanych wizualizacji przyszła pora na właściwą implementację projektu w języku Python. Omówienie tego procesu zostało podzielone na tematyczne podrozdziały, w celu jak najlepszego uporządkowania prezentacji postępu prac.

4.1. Struktura projektu



Rys. 4.1: Zrzut ekranu przedstawiający hierarchię plików wchodzących w skład projektu inżynierskiego

Głównym elementem w hierarchii plików jest katalog `src`. Na równi z nim znajdują się pliki `.gitignore` oraz `requirements.txt`, które służą kolejno do ułatwienia pracy z repozytorium kodu poprzez określenie plików niepodlegających śledzeniu przez narzędzie kontroli wersji oraz do usprawnienia procesu dystrybucji kodu dla osób zainteresowanych uruchomieniem projektu na

własnym komputerze poprzez kompletny spis modułów wykorzystanych w projekcie. W folderze *src* umieszczono trzy kluczowe podfoldery - *data*, *viz* oraz *webapp*. Pierwszy z nich zawiera wszystkie pliki z danymi, na podstawie których generowane są wizualizacje. Katalog *viz* przechowuje skrypty generujące pliki wizualizacyjne .html, a także garść funkcji pomocniczych (*utils.py*) oraz spersonalizowane makro kodów HTML i CSS tworzące legendę mapy(*legend_template.txt*). Folder *webapp* zawiera kod prostego serwera webowego służącego do hostowania stron z wizualizacjami i wykonywania cyklicznych zadań aktualizacji danych oraz pliki źródłowe stron w formacie .html. W katalogach zawierających kod w języku Python można również zauważać foldery o nazwie *_pycache_*, gdzie znajduje się kod bitowy wygenerowany przez interpreter na podstawie skryptów .py [11], aczkolwiek interakcja z nim nie była potrzebna w trakcie realizacji projektu, dlatego dalszy opis zostanie pominięty (rys. 4.1).

4.2. Skrypty generujące wizualizacje

4.2.1. Najważniejsze wykorzystane biblioteki

folium

Narzędzie, które stoi u podstaw całego projektu inżynierskiego to pythonowy moduł *folium*. Stworzono go z myślą o połączeniu siły języka Python w obszarze analizy i przetwarzania danych z potęgą biblioteki *leaflet* języka JavaScript w zakresie tworzenia interaktywnych map. Innymi słowy, moduł ten jest tzw. "wrapperem", który efektywnie ułatwia programistom języka Python korzystanie z dobrodziesztw biblioteki *leaflet* bez konieczności znajomości języka JavaScript. Ogromną zaletą *folium* jest wsparcie społeczności oraz rozszerzalność gwarantowana przez liczne wtyczki [3], spośród których kilka wykorzystano w opisywanym projekcie.

pandas oraz *geopandas*

Moduł *pandas* został stworzony w celu wsparcia procesu praktycznej analizy i przetwarzania danych w języku Python na wysokim poziomie abstrakcji. Biblioteka wspiera również wczytywanie i zapisywanie danych o formatach takich jak CSV, pliki tekstowe, Microsoft Excel czy bazy danych SQL. Podstawowymi założeniami narzędzia jest jego darmowość, elastyczność i łatwość w użyciu [6]. *Geopandas* jest jego rozszerzeniem, które dostarcza rozwiązań ułatwiających pracę ze stricte geoprzestrzennymi zbiorami danych poprzez np. wsparcie importu z plików GeoJSON [4].

branca

Inicjatywa powiązana z modułem *folium*, która skupia się na aspektach wizualizacji kartograficznych niezwiązanych bezpośrednio z mapami. Dostarcza między innymi oprogramowanie służące do tworzenia map kolorów oraz makr, które można dodać do wizualizacji [15].

numpy

Projekt mający na celu ułatwić i zoptymalizować obliczenia numeryczne w języku Python. Zawiera oprogramowanie służące do m.in.: obliczeń związanych z algebra liniową, transformatą Fouriera, macierzami oraz liczbami losowymi [5].

4.2.2. Schemat działania skryptów generujących wizualizacje

Pomimo faktu, iż każdy z programów generuje wizualizację o specyfice różniącej się w mniejszym lub większym stopniu od specyfiki innych, to możliwe jest wyróżnienie pewnego uniwersalnego szablonu ich działania:

1. Import i wstępne przetwarzanie danych źródłowych.
2. Inicjalizacja obiektu mapy z modułu *folium*.
3. Stworzenie obiektów odpowiadających za zawartość mapy, przekazanie im przetwarzonych w fazie pierwszej danych źródłowych, a ostatecznie powiązanie ich z obiektem mapy.
4. Zapisanie gotowego obiektu mapy do określonego pliku .html.

4.2.3. Legenda mapy

Moduł *folium* jest narzędziem znajdującym się w fazie rozwoju, dlatego nie posiada on jeszcze pewnych przydatnych rozwiązań, czego naturalną konsekwencją jest zjawisko zgłaszania, a także implementacji ich poprzez osoby trzecie. Niestety istniejące w chwili obecnej (5.12.2020) oprogramowanie legendy dołączanej do mapy dostępne jest jedynie pod postacią mapy kolorów. Nie sprostało to wymaganiom klarowności i skalowalności opisu prezentowanych danych. Najlepszym rozwiązaniem tego problemu okazałaby się implementacja własnego szablonu wielokrotnego użytku w języku HTML, CSS i JavaScript, gdyby nie fakt, że pewien użytkownik wspierający projekt już je dostarczył [24]. W znalezionym szablonie dokonano kilku zmian i jego stan obecny przedstawia listing 4.1.

Listing 4.1: Plik legend_template.txt zawierający szablon do tworzenia legendy

```

1  {% macro html(this, kwargs) %} 
2  <!doctype html>
3  <html lang="en">
4  <head>
5      <meta charset="utf-8">
6      <meta name="viewport" content="width=device-width, initial-scale=1">
7      <link rel="stylesheet" href="//code.jquery.com/ui/1.12.1/themes/base/jquery-ui.css">
8      <script src="https://code.jquery.com/jquery-1.12.4.js"></script>
9      <script src="https://code.jquery.com/ui/1.12.1/jquery-ui.js"></script>
10     <script>
11         $( function() {
12             $(".maplegend").draggable({
13                 start: function (event, ui) {
14                     $(this).css({
15                         right: "auto",
16                         top: "auto",
17                         bottom: "auto"
18                     });
19                 }
20             });
21         });
22     </script>
23 </head>
24 <body>
25     <div id='maplegend' class='maplegend'
26         style='position: absolute; z-index:9999; border:2px solid grey; background-color:rgba
27             (255, 255, 255, 0.8);
28             border-radius:6px; padding: 10px; font-size:14px; right: 20px; bottom: 20px;'>
29         <div class='legend-title'>Legend (draggable!)</div>
30         <div class='legend-scale'>
31             <ul class='legend-labels'>
32
33             </ul>
34         </div>
35     </div>
36 </body>
37 </html>
38
39 <style type='text/css'>
```

```

40 .maplegend .legend-title {
41   text-align: left;
42   margin-bottom: 5px;
43   font-weight: bold;
44   font-size: 90%;
45 }
46 .maplegend .legend-scale ul {
47   margin: 0;
48   margin-bottom: 5px;
49   padding: 0;
50   float: left;
51   list-style: none;
52 }
53 .maplegend .legend-scale ul li {
54   font-size: 80%;
55   list-style: none;
56   margin-left: 0;
57   line-height: 18px;
58   margin-bottom: 2px;
59 }
60 .maplegend ul.legend-labels li span {
61   display: inline-block;
62   height: 16px;
63   width: 30px;
64   margin-right: 5px;
65   margin-left: 0;
66   border: 1px solid #999;
67 }
68 .maplegend .legend-source {
69   font-size: 80%;
70   color: #777;
71   clear: both;
72 }
73 .maplegend a {
74   color: #777;
75 }
76 </style>
77 {%
  endmacro %}

```

Ponieważ projekt w swoich założeniach stawia na automatyzację procesu generowania wizualizacji, należało napisać odpowiednią funkcję, której zadaniem było wczytanie szablonu i wypełnienie go danymi określonymi dla konkretnego przypadku. Funkcja zawarta w listingu 4.2 generuje legendę z tytułem i skalowalną zawartością, niezależnie czy znajdzie się na niej jedna kategoria danych czy więcej. Ma za zadanie zwrócić kod HTML/CSS/JavaScript w obiekcie String, którego dalsza obsługa odbywa się w poszczególnych skryptach generujących wizualizacje. Końcowy efekt działania wygenerowanego kodu można zaobserwować na rys. 4.2.

Listing 4.2: Funkcja create_legend z pliku utils.py

```

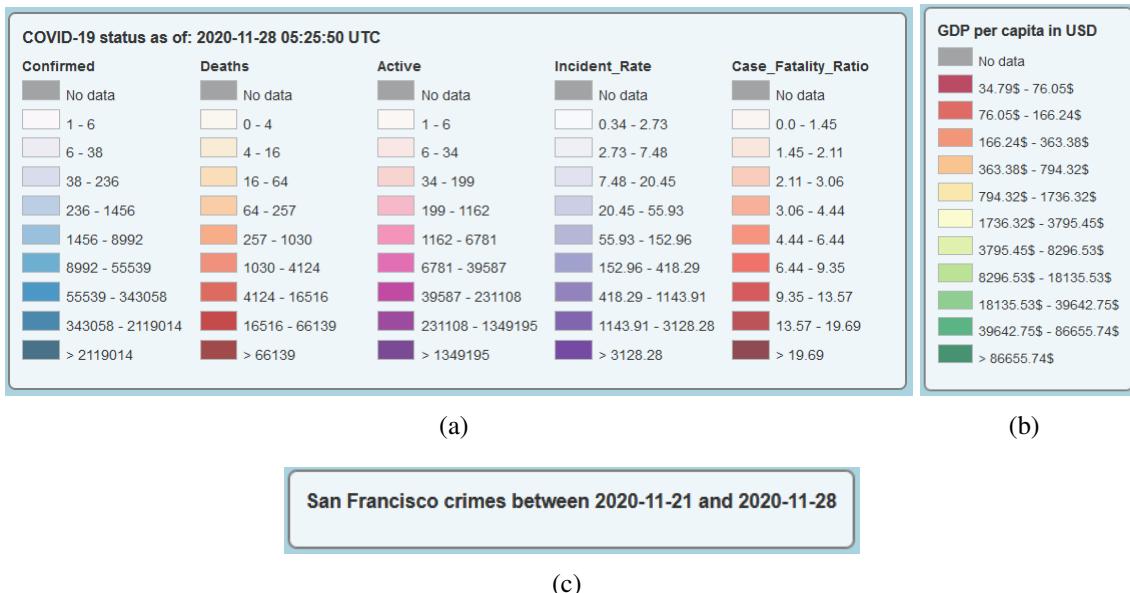
1 import os
2
3 script_dir_path = os.path.dirname(os.path.realpath(__file__))
4
5 # Funkcja przyjmuje przyjmuje tytuł legendy oraz dane, które mają się na niej znaleźć
6 def create_legend(caption=None, legend_labels=None) -> str:
7     # Wczytywanie zawartości pliku szablonu
8     file = open(os.path.join(script_dir_path, 'legend_template.txt'), 'r')
9     lines = file.readlines()
10    file.close()
11
12    # Nadpisanie domyślnej nazwy legendy
13    if caption is not None:
14        legend_title = "<div class='legend-title'>" + caption + "</div>\n"
15        lines[lines.index("<div class='legend-title'>Legend (draggable!)</div>\n")] =
16            legend_title
17
18    # Opuszczenie funkcji w przypadku, gdy zawartość legendy stanowi tylko jej tytuł
19    if legend_labels is None:
20        separator = ''
21        return separator.join(lines)
22
23    # Logika obsługi dla wielu kategorii danych - lista w legend_labels
24    if any(isinstance(value, dict) for value in legend_labels.values()):

```

```

24     css_index = lines.index("<style type='text/css'>\n") + 1
25     # Szablon dekoratora, który tworzy odpowiednio wyskalowany widok legendy
26     css_decorator = ''
27         .legend-labels {{
28             columns: {};
29             -webkit-columns: {};
30             -moz-columns: {};
31         }}
32     '''.format(len(legend_labels), len(legend_labels), len(legend_labels))
33     lines.insert(css_index, css_decorator)
34     label_index = lines.index("  <ul class='legend-labels'>\n") + 1
35     for _, (category, legend) in enumerate(legend_labels.items()):
36         lines.insert(label_index, "    <li><b>" + category + "</b></li>\n")
37         label_index += 1
38         for _, (k, v) in enumerate(legend.items()):
39             legend_label = "      <li><span style='background:" + k + ";opacity:0.7;'></span>
40             ↪ >" + v + "</li>\n"
41             lines.insert(label_index, legend_label)
42             label_index += 1
43
44     # Logika obsługi dla jednej kategorii danych - słownik w legend_labels
45     else:
46         label_index = lines.index("  <ul class='legend-labels'>\n") + 1
47         for _, (k, v) in enumerate(legend_labels.items()):
48             legend_label = "      <li><span style='background:" + k + ";opacity:0.7;'></span>
49             ↪ >" + v + "</li>\n"
50             lines.insert(label_index, legend_label)
51             label_index += 1
52
52     separator = ''
52     return separator.join(lines)

```



Rys. 4.2: Zestawienie legend z danymi w różnej skali: (a) pięć kategorii danych, (b) jedna kategoria danych, (c) brak danych

4.2.4. Wizualizacja PKB per capita (lata 1960-2019)

U podstaw omawianego podprojektu leży wykorzystanie wtyczki do modułu *folium* o nazwie *TimeSliderChoropleth*. Dostarcza ona rozwiązania programistyczne umożliwiające powiązanie danych tworzących pojedyncze kartogramy z tzw. znakami czasowymi (ang. *timestamp*), w efekcie umożliwiając interaktywną prezentację danych zmieniających się na przestrzeni czasu. Kluczowymi parametrami inicjalizacyjnymi, które przyjmuje konstruktor klasy *TimeSliderChoropleth*, są atrybuty *data* oraz *styledict*. Pierwszy z nich jest informacją o kształcie regionów

geograficznych w formacie GeoJSON, a drugi pythonowym słownikiem stylów zawierającym dane wpisujące się w schemat z listingu 4.3.

Listing 4.3: Szablon zawartości słownika stylów

```

1 style_dict = {
2     'indeks_odpowiadający_danemu_obszarowi_geograficznemu_w_parametrze_data': {
3         'czas_wyrażony_w_milisekundach' : {
4             'color': 'kolor_przypisany_do_danego_regionu_w_danym_czasie',
5             'opacity': 'wartość_przezroczystość_wyświetlanego_koloru'
6         }
7     ...
8 }
9 ...
10 }
```

Głównym problemem, jaki wystąpił w trakcie realizacji tego podprojektu okazała się niepełna kompatybilność kodów ISO_A3 pomiędzy zbiorem geometrii granic GeoJSON, a danymi liczbowymi CSV dot. PKB. Ponieważ niekompatybilność dotyczyła całkiem sporej liczby państw, zdecydowano się na rozwiązań polegające na pominięciu rekordów, których kod ISO_A3 wynosił -99 (oznaczenie braku kodu). Należało również wykonać kilka operacji związanych z teorią zbiorów, a konkretnie wykorzystać operatory iloczynu i negacji (rozwiązań dostarczone przez moduł *pandas*), aby upewnić się, że w obu obiektach mamy do czynienia z tym samym zestawem danych regionalnych.

Rezultaty kodu generującego plik z wizualizacją (listing 4.4) przedstawiono na rys. 4.3

Listing 4.4: Zawartość pliku gdp_viz.py

```

1 import folium
2 import time
3 import os
4 import utils
5 import pandas as pd
6 import geopandas as gpd
7 import numpy as np
8 import branca.colormap as cm
9 from datetime import datetime
10 from folium.plugins import TimeSliderChoropleth
11 from branca.element import Template, MacroElement
12
13 script_dir_path = os.path.dirname(os.path.realpath(__file__))
14 pd.set_option('display.max_rows', None)
15
16 def create_gdp_viz():
17     """
18     Załadowanie i wstępne przetworzenie pliku GeoJSON
19     """
20     geojson_path = os.path.normpath(os.path.join(script_dir_path, '..', 'data', 'borders_geo.
21         ↪ json'))
21     world_geojson = gpd.read_file(geojson_path)
22     world_geojson.drop(columns=['ISO_A2', 'ADMIN'], inplace=True)
23     world_geojson.drop(world_geojson[world_geojson['ISO_A3'] == '-99'].index, inplace=True)
24     country_list = world_geojson['ISO_A3'].tolist()
25
26     """
27     Załadowanie i wstępne przetworzenie pliku z danymi o PKB per capita
28     """
29     # Załadowanie danych do obiektu DataFrame z modułu pandas
30     df_GDP_path = os.path.normpath(os.path.join(script_dir_path, '..', 'data', 'GDP_per_capita_world_data.csv'))
31     df_GDP = pd.read_csv(df_GDP_path, index_col='Country Code', skiprows=4)
32
33     # Usunięcie pustej kolumny z danymi z roku 2020
34     df_GDP.drop(labels='2020', axis=1, inplace=True)
35
36     # Operacje mające na celu zapewnienie zgodności danych pomiędzy obiektem geometrii granic
37         ↪ oraz obiektem zawierającym wizualizowane dane
38     csv_country_list = df_GDP.index.tolist()
39     country_list = list(set(country_list).intersection(csv_country_list))
40     df_GDP.drop(df_GDP[~df_GDP.index.isin(country_list)].index, inplace=True)
        ↪ inplace=True)
    world_geojson.drop(world_geojson[~world_geojson['ISO_A3'].isin(country_list)].index,
```

```

41     country_list.sort()
42
43     # Stworzenie słownika państw w celu zmapowania identyfikatorów
44     country_dict = {k: v for v, k in enumerate(country_list)}
45     world_geojson['country_id']=world_geojson['ISO_A3'].map(country_dict)
46
47     # Policzenie minimalnej i maksymalnej wartości z całego zbioru danych
48     min_GDP_val, max_GDP_val = df_GDP[df_GDP.columns[4:]].min().min(), df_GDP[df_GDP.columns
49         ↪ [4:]].max().max()
50
51     # Stworzenie listy z kodami kolorów odpowiadających poszczególnym przedziałom wartości na
52         ↪ mapie
53     color_list = [
54         '#808080', '#A50026', '#D73027', '#F46D43', '#FDAE61', '#FEE08B', '#FFFFBF', '#D9EF8B', '#
55             ↪ A6D96A', '#66BD63', '#1A9850', '#006837'
56     ]
57
58     # Stworzenie przestrzeni geometrycznej na bazie wyliczonych ekstremów przy użyciu modułu
59         ↪ numpy
60     bins = np.geomspace(min_GDP_val, max_GDP_val, 12)
61
62     # Zamiana wartości Nan na -1 w rekordach, w których występują puste komórki
63     df_GDP.fillna(-1, inplace=True)
64
65     # Dodanie przedziału <-1, 0) do przestrzeni geometrycznej, zaliczą się do niej wyłącznie
66         ↪ rekordy z wartością -1, czyli brak danych
67     bins = np.insert(bins, 0, -1.)
68     bins = bins.tolist()
69
70     # Dodanie kolumny 'color_[rok]' do obiektu df_GDP
71     year = 1960
72     while year <= 2019:
73         pasted_col_id = df_GDP.columns.get_loc(str(year)) + 1
74         col_value = pd.cut(df_GDP[str(year)], bins, include_lowest=True, labels =
75             ['#808080', '#A50026', '#D73027', '#F46D43', '#FDAE61', '#FEE08B', '#FFFFBF', '#D9EF8B', '#
76                 ↪ A6D96A', '#66BD63', '#1A9850', '#006837'
77         ])
78         df_GDP.insert(loc=pasted_col_id, column='color_'+str(year), value=col_value)
79         year += 1
80     print(df_GDP)
81
82     ...
83     Inicjalizacja mapy
84     ...
85     map_GDP = folium.Map(location=[0, 0], zoom_start=4, max_bounds=True, min_zoom=3)
86
87     ...
88     Stworzenie zawartości mapy i dodanie jej do obiektu map_GDP
89     ...
90     # Stworzenie odpowiednio sformatowanego słownika stylów, który zostanie przekazany do
91         ↪ obiektu TimeSliderChoropleth (interaktywny kartogram z suwakiem)
92     gdp_dict = {}
93     for country_code in df_GDP.index.tolist():
94         country_id = str(country_dict[country_code])
95         gdp_dict[country_id] = {}
96         year = 1960
97         while year <= 2019:
98             dt_obj = datetime(year=year, month=12, day=31)
99             year_in_ms = str(time.mktime(dt_obj.timetuple()))
100            color_hex = df_GDP.at[country_code, 'color_'+str(year)]
101            gdp_dict[country_id][year_in_ms] = {'color':color_hex, 'opacity':0.7}
102            year += 1
103
104        # Stworzenie obiektu kartogramu
105        choropleth = TimeSliderChoropleth(
106            world_geojson.set_index('country_id').to_json(),
107            styledict=gdp_dict
108        )
109        choropleth.add_to(map_GDP)

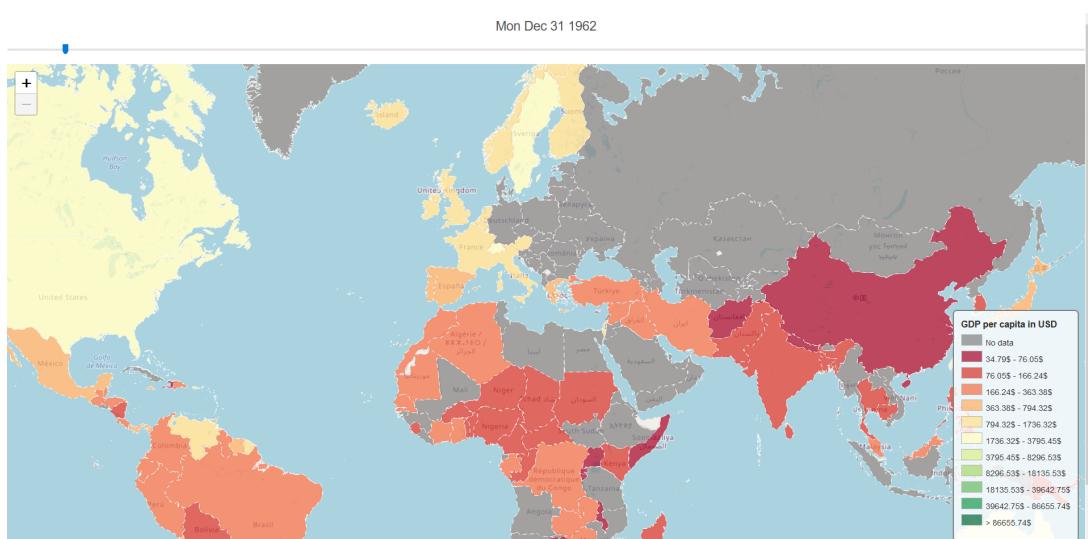
110        # Stworzenie legendaru do mapy
111        legend_labels_dict = {}
112        i = 0
113        for color in color_list:
114            if i == 0:
115                legend_labels_dict[color_list[i]] = 'No data'

```

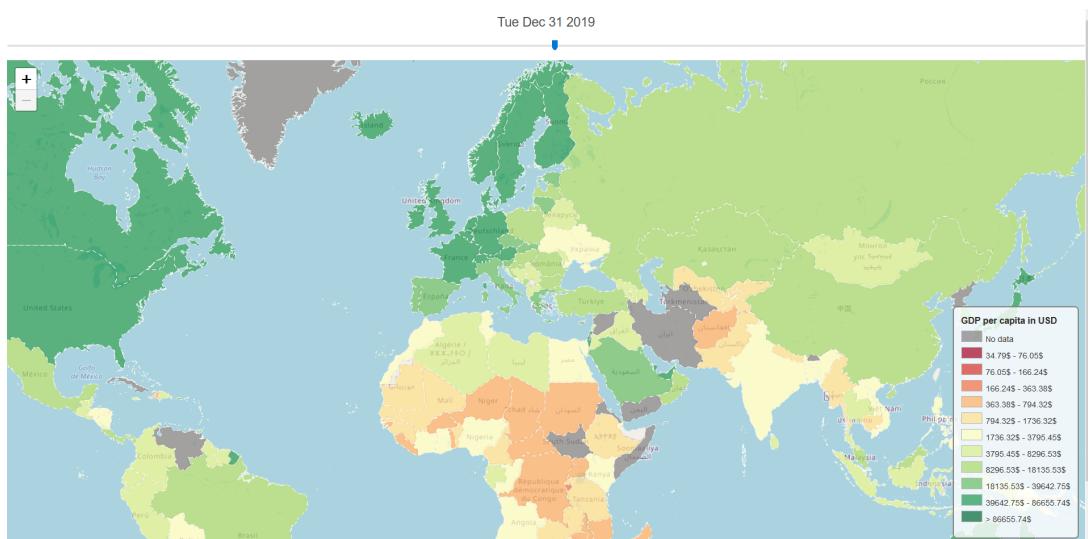
```

110     elif i == len(color_list) - 1:
111         legend_labels_dict[color_list[i]] = '> ' + str(round(bins[i], 2)) + '$'
112         break
113     else:
114         legend_labels_dict[color] = str(round(bins[i], 2)) + '$' + ' - ' + str(round(bins[
115             ↪ i+1], 2)) + '$'
116         i += 1
117     template = utils.create_legend(caption='GDP per capita in USD', legend_labels=
118         ↪ legend_labels_dict)
119     macro = MacroElement()
120     macro._template = Template(template)
121     map_GDP.get_root().add_child(macro)
122
123 ...
124 Zapisanie ukończonej wizualizacji do odpowiedniego pliku
125
map_GDP.save(os.path.join(script_dir_path, '...', 'webapp', 'templates', 'GDP_viz.html'))
print('Successfully created the GDP viz!')

```



(a)



(b)

Rys. 4.3: Zestawienie rezultatów wizualizacji dla lat: (a) 1962, (b) 2019. Zauważalna jest tendencja do większych braków w danych sprzed kilkudziesięciu lat. Przyczyn tego zjawiska jest wiele, do niektórych z nich można zaliczyć kwestię nieistnienia naówczas niektórych z państw obecnie funkcjonujących, a także brak publikacji tego typu danych ze względów politycznych

4.2.5. Wizualizacja obecnego stanu epidemii COVID-19

W przypadku drugiego podprojektu kluczowym założeniem było napisanie programu, który stworzy wizualizację będącą w stanie przedstawić kilka kategorii danych (omówienie aktualizacji tychże danych znajduje się w rozdziale 4.3) Do jego zrealizowania wykorzystano mechanizm zwany *FeatureGroup* - klasę dostępną w module *folium*. Umożliwia one logiczne powiązanie w grupy określonych obiektów zawartości mapy, co w konsekwencji prowadzi do możliwości przełączania warstw mapy w celu zmiany wyświetlanej kategorii danych.

Pierwszy z problemów napotkanych podczas pisania kodu programu przypominał analogiczną komplikację, napotkaną przy okazji pracy nad poprzednim podprojektem - niepełna kompatybilność, tym razem nie w kodach ISO_A3 (takowych zbiór danych epidemiologicznych nie posiada), lecz w samych nazwach państw, które dlatego musiały zostać wykorzystane jako klucz powiązujący dane wektorowe geometrii granic oraz wizualizowane dane liczbowe. Niekompatybilność w nazewnictwie nie sięgała jednak tak daleko, jak w przypadku wizualizacji PKB per capita, dlatego możliwa była ręczna korekta danych.

Ponieważ obiekty odpowiadające w bibliotece *folium* za tworzenie kartogramów nie dziedziczą z klasy *Layer* [3], nie da się zastosować ich przy jednoczesnym wykorzystaniu klasy *FeatureGroup*. Zamiast nich utworzono obiekty klasy *GeoJson*, których zastosowanie umożliwiło bardzo intuicyjną i szybką implementację funkcjonalności "popup'ów" informacyjnych, pojawiających się po kliknięciu na dany obszar geograficzny, która inaczej nie byłaby możliwa dla klasy *Choropleth*.

Ostatnie duże wyzwanie, które zostało już dokładniej omówione w rozdziale 4.2.3, wiązało się ze stworzeniem legendy dostosowanej do specyfiki wizualizacji. Pomijając poruszoną kwestię szablonu w HTML/CSS/JavaScript, również kod w Pythonie wymagał odpowiedniego dostosowania struktury danych przekazywanej do funkcji generującej. Ze względu na wielowymiarowość danych należało stworzyć i przekazać odpowiednio sformatowaną listę słowników.

Rezultaty uzyskane z kodu przedstawionego na listingu 4.5 zaprezentowano na rysunkach 4.4, 4.5, 4.6 w postaci przykładowych widoków wizualizacji.

Listing 4.5: Funkcja `create_covid_viz` i jej zależności z pliku `covid_viz.py`

```

1 import folium
2 import os
3 import utils
4 import glob
5 import json
6 import pandas as pd
7 import numpy as np
8 import geopandas as gpd
9 import branca.colormap as cm
10 from branca.element import MacroElement
11 from jinja2 import Template
12
13 # Deklaracja istotnych ścieżek w strukturze projektu
14 script_dir_path = os.path.dirname(os.path.realpath(__file__))
15 data_dir_path = os.path.join(script_dir_path, '..', 'data')
16 geojson_path = os.path.normpath(os.path.join(script_dir_path, '..', 'data', 'borders_geo.json'
17                                     ))
18
19 # Pobranie nazwy pliku z danymi epidemiologicznymi
20 try:
21     newest_dataset = glob.glob(os.path.join(data_dir_path, 'covid*'))[0].rsplit('/', 1)[-1]
22 except:
23     print('No Covid dataset found in the data folder')
24
25 # Deklaracja palet kolorów wykorzystanych w wizualizacji
26 color_dict = {
27     'Confirmed': ['#808080', '#fff7fb', '#ece7f2', '#d0d1e6', '#a6bddb', '#74a9cf', '#3690c0', '#0570
28                                     ↪ b0', '#045a8d', '#023858'],
29     'Deaths': ['#808080', '#fff7ec', '#fee8c8', '#fdd49e', '#fdbb84', '#fc8d59', '#ef6548', '#d7301f'
29                                     ↪ , '#b30000', '#7f0000'],
30     'Active': ['#808080', '#fff7f3', '#fde0dd', '#fcc5c0', '#fa9fb5', '#f768a1', '#dd3497', '#ae017e'
30                                     ↪ , '#7a0177', '#49006a'],
31 }
```

```

29     'Incident_Rate': ['#808080', '#fcfbfd', '#efedf5', '#dadaeb', '#bcbddc', '#9e9ac8', '#807dba', '#6a51a3', '#54278f', '#3f007d'],
30     'Case_Fatality_Ratio': ['#808080', '#ffff5f0', '#fee0d2', '#fcbb1', '#fc9272', '#fb6a4a', '#ef3b2c', '#cb181d', '#a50f15', '#67000d']
31 }
32
33 def create_covid_viz():
34     """
35     Załadowanie i wstępne przetworzenie pliku GeoJSON
36     """
37     world_geojson = gpd.read_file(geojson_path)
38     world_geojson.drop(columns=['ISO_A2'], inplace=True)
39
39     """
40     Załadowanie i wstępne przetworzenie danych epidemiologicznych
41     """
42
43     # Load the COVID-19 data
44     df_covid = pd.read_csv(os.path.join(data_dir_path, newest_dataset))
45     timestamp = df_covid['Last_Update'][0]
46
47     # Zmiana niekompatybilnych nazw państw
48     df_covid.replace(to_replace={'Country_Region' : 'US'}, value='United States of America',
49                      inplace=True)
49     df_covid.replace(to_replace={'Country_Region' : 'Bahamas'}, value='The Bahamas', inplace=
50                      True)
50     df_covid.replace(to_replace={'Country_Region' : 'Congo (Brazzaville)'}, value='Republic of
51                      Congo', inplace=True)
51     df_covid.replace(to_replace={'Country_Region' : 'Congo (Kinshasa)'}, value='Democratic
52                      Republic of the Congo', inplace=True)
52     df_covid.replace(to_replace={'Country_Region' : 'Taiwan*'}, value='Taiwan', inplace=True)
53     df_covid.replace(to_replace={'Country_Region' : "Cote d'Ivoire"}, value='Ivory Coast',
54                      inplace=True)
54     df_covid.replace(to_replace={'Country_Region' : "Czechia"}, value='Czech Republic',
55                      inplace=True)
55     world_geojson.replace(to_replace={'ADMIN' : 'Macedonia'}, value='North Macedonia', inplace
56                      =True)
56
57     # Zmiana nazwy kolumny 'ADMIN' z obiektu GeoJSON na identyczną do wykorzystanej w danych
58     # epidemiologicznych, w celu ułatwienia mapowania danych
59     Change the name of 'ADMIN' column in the geojson DF to match the one in COVID DF
59     world_geojson.rename(columns={'ADMIN': 'Country_Region'}, inplace=True)
60
61     # Agregacja danych dla krajów posiadających informacje rozbite na ich poszczególne regiony
62     df_covid_agg = df_covid.groupby('Country_Region').agg({'Confirmed': 'sum',
63                                         'Deaths': 'sum',
64                                         'Recovered': 'sum',
65                                         'Active': 'sum',
66                                         'Incident_Rate': 'mean',
67                                         'Case_Fatality_Ratio': 'mean'})
68     world_geojson = world_geojson.sort_values('Country_Region').reset_index(drop=True)
69
70     # Połączenie danych z obiektu GeoJSON z danymi z obiektu DataFrame
71     df_covid_joined = df_covid_agg.merge(world_geojson, how='right', on='Country_Region')
72
73     # Zliczanie minimalnych i maksymalnych wartości dla każdej kategorii wizualizacji
74     min_dict, max_dict = {}, {}
75     column_names = ['Confirmed', 'Deaths', 'Active', 'Incident_Rate', 'Case_Fatality_Ratio']
76     for name in column_names:
77         min_dict[name] = min(df_covid_joined[name])
78         max_dict[name] = max(df_covid_joined[name])
79
80     # Zamiana wartości pustych komórek na -1 w celu ustalonyzowania obliczeń
81     df_covid_joined.fillna(-1, inplace=True)
82
83     # Dodanie danych epidemiologicznych do pliku GeoJSON, koniecznych dla przyszłego wyś
83     # wietlania wyskakujących okienek
84     Add the data columns to geo json for future popup displaying
85     world_geojson = world_geojson.assign(Confirmed=df_covid_joined['Confirmed'],
86                                         Deaths=df_covid_joined['Deaths'],
87                                         Active=df_covid_joined['Active'],
88                                         Incident_Rate=df_covid_joined['Incident_Rate'],
89                                         Case_Fatality_Ratio=df_covid_joined[
90                                         Case_Fatality_Ratio'])
90     print(world_geojson)
91
92     # Modyfikacja kolumny indeksującej w obiekcie DataFrame

```

```

93 df_covid_joined.set_index('Country_Region', inplace=True)
94
95 # Stworzenie list przestrzeni geometrycznych dla poszczególnych kategorii wizualizacji w
#       → celu przypisania koloru do danych
96 colormap_dict = {}
97 bins = []
98 for name in column_names:
99     # Rozwiązywanie problemu nieakceptacji przez przestrzeń geometryczną zer w sekwencji
#       → liczbowej
100    tmp_min = min_dict[name]
101    if min_dict[name] < 1:
102        min_dict[name] = 1
103
104    # Stworzenie przestrzeni geometrycznej i uzupełnienie jej o przedział dla brakujących
#       → danych
105    inner_bins = np.geomspace(start=min_dict[name], stop=max_dict[name], num=10)
106    min_dict[name] = tmp_min
107    inner_bins = np.delete(inner_bins, 0)
108    inner_bins = np.insert(inner_bins, 0, min_dict[name])
109    inner_bins = np.insert(inner_bins, 0, -1.)
110    inner_bins = inner_bins.tolist()
111
112    # Zaokrąglenie wartości krańców przedziałów dla tych kategorii wizualizacji, których
#       → dane są liczbami całkowitymi
113    if name in ['Confirmed', 'Deaths', 'Active']:
114        inner_bins = [int(round(bin, 0)) for bin in inner_bins]
115    else:
116        inner_bins = [round(bin, 2) for bin in inner_bins]
117
118    # Stworzenie mapy kolorów oraz dodanie zmapowanych za jej pomocą wartości do
#       → odpowiednich kolumn
119    bins.append(inner_bins)
120    colormap_dict[name] = cm.StepColormap(colors=color_dict[name], index=inner_bins, vmin=
#       → min_dict[name], vmax=max_dict[name])
121    df_covid_joined[name+'_color'] = df_covid_joined[name].map(lambda x: colormap_dict[
#       → name].rgb_hex_str(x))
122
123 ...
124 Inicjalizacja mapy
125 ...
126 map_covid = folium.Map(location=[0, 0], zoom_start=4, max_bounds=True, tiles=None)
127 base_map = folium.FeatureGroup(name='Basemap', overlay=True, control=False)
128 folium.TileLayer(min_zoom=3, tiles='OpenStreetMap').add_to(base_map)
129 base_map.add_to(map_covid)
130
131 ...
132 Stworzenie zawartości mapy
133 ...
134 # Stworzenie obiektów FeatureGroup pełniących rolę swego rodzaju warstw mapy, pomiędzy którymi możliwe jest przełączanie się.
135 feature_groups = []
136 for category, _ in color_dict.items():
137     group = folium.FeatureGroup(category, overlay=False)
138     feature_groups.append(group)
139
140 # Deklaracja kartogramów z wykorzystaniem obiektu folium.GeoJSON, okienek wyskakujących
#       → przy kliknięciu na dany region oraz dodanie obu obiektów do odpowiedniej warstwy na
#       → mapie
141 choropleth_confirmed = folium.GeoJson(data=world_geojson,
142                                         zoom_on_click=False,
143                                         name='Confirmed_Cases',
144                                         style_function=lambda x: {
145                                             'fillColor': df_covid_joined['Confirmed_color'][
#       → x['properties']['Country_Region']],
146                                             'fillOpacity': 0.7,
147                                             'color': 'black',
148                                             'weight': 0.5
149                                         }).add_to(feature_groups[0])
150 popup_confirmed = folium.GeoJsonPopup(fields=['Country_Region', 'Confirmed'], labels=False
#       → )
151 popup_confirmed.add_to(choropleth_confirmed)
152
153 choropleth_deaths = folium.GeoJson(data=world_geojson,
154                                         zoom_on_click=False,
155                                         name='Deaths',
156                                         style_function=lambda x: {

```

```

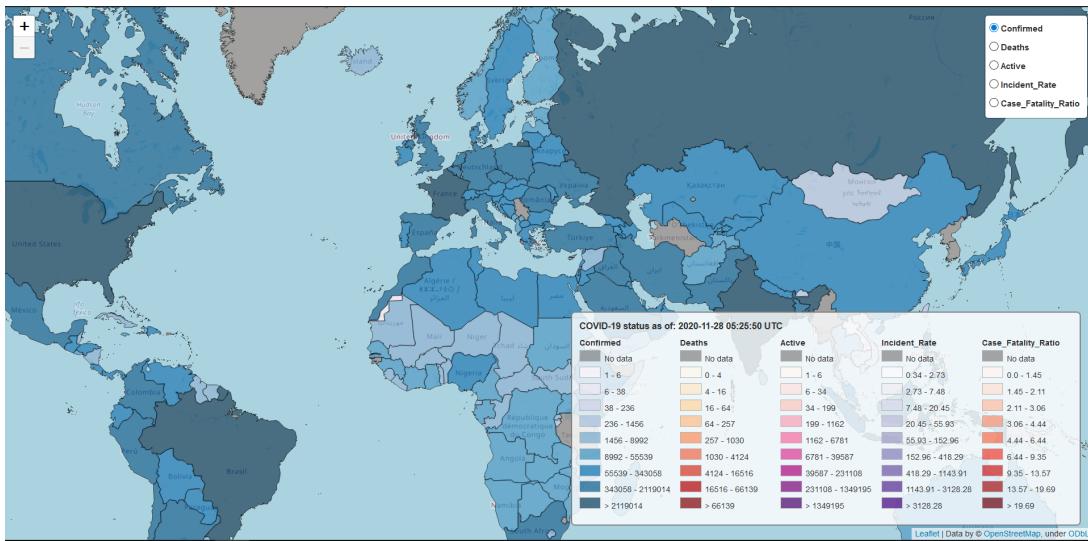
157                               'fillColor': df_covid_joined['Deaths_color'][x['
158                                     ↪ properties']][['Country_Region']],
159                               'fillOpacity': 0.7,
160                               'color': 'black',
161                               'weight': 1
162                         }).add_to(feature_groups[1])
163 popup_deaths = folium.GeoJsonPopup(fields=['Country_Region', 'Deaths'], labels=False)
164 popup_deaths.add_to(choropleth_deaths)
165
166 choropleth_active = folium.GeoJson(data=world_geojson,
167                                     zoom_on_click=False,
168                                     name='Active Cases',
169                                     style_function=lambda x: {
170                                         'fillColor': df_covid_joined['Active_color'][x['
171                                             ↪ properties']][['Country_Region']],
172                                         'fillOpacity': 0.7,
173                                         'color': 'black',
174                                         'weight': 1
175                         }).add_to(feature_groups[2])
176 popup_active = folium.GeoJsonPopup(fields=['Country_Region', 'Active'], labels=False)
177 popup_active.add_to(choropleth_active)
178
179 choropleth_incident_rate = folium.GeoJson(data=world_geojson,
180                                              zoom_on_click=False,
181                                              name='Incident Rate',
182                                              style_function=lambda x: {
183                                                 'fillColor': df_covid_joined['
184                                     ↪ Incident_Rate_color'][x['properties'
185                                     ↪ ][['Country_Region']],
186                                                 'fillOpacity': 0.7,
187                                                 'color': 'black',
188                                                 'weight': 1
189                         }).add_to(feature_groups[3])
190 popup_incident_rate = folium.GeoJsonPopup(fields=['Country_Region', 'Incident_Rate'],
191                                              ↪ labels=False)
192 popup_incident_rate.add_to(choropleth_incident_rate)
193
194 choropleth_case_fatality_ratio = folium.GeoJson(data=world_geojson,
195                                              zoom_on_click=False,
196                                              name='Case Fatality Ratio',
197                                              style_function=lambda x: {
198                                                 'fillColor': df_covid_joined['
199                                     ↪ Case_Fatality_Ratio_color'][x[
200                                         ↪ 'properties'][['Country_Region'
201                                         ↪ ]],
202                                                 'fillOpacity': 0.7,
203                                                 'color': 'black',
204                                                 'weight': 1
205                         }).add_to(feature_groups[4])
206 popup_case_fatality_ratio = folium.GeoJsonPopup(fields=['Country_Region', '
207                                         ↪ Case_Fatality_Ratio'], labels=False)
208 popup_case_fatality_ratio.add_to(choropleth_case_fatality_ratio)
209
210 # Stworzenie legendy do mapy
211 legend_str_dict = {}
212 for i, (k, v) in enumerate(color_dict.items()):
213     legend_labels_dict = {}
214     j = 0
215     for color in v:
216         if j == 0:
217             legend_labels_dict[color] = 'No data'
218         elif j == len(v) - 1:
219             legend_labels_dict[color] = '> ' + str(bins[i][j])
220             break
221         else:
222             legend_labels_dict[color] = str(bins[i][j]) + ' - ' + str(bins[i][j+1])
223             j += 1
224     legend_str_dict[k] = legend_labels_dict
225
226 template = utils.create_legend(caption='COVID-19 status as of: ' + str(timestamp) + ' UTC',
227                                ↪ legend_labels=legend_str_dict)
228 macro = MacroElement()
229 macro._template = Template(template)
230 map_covid.get_root().add_child(macro)
231
232 for feature_group in feature_groups:

```

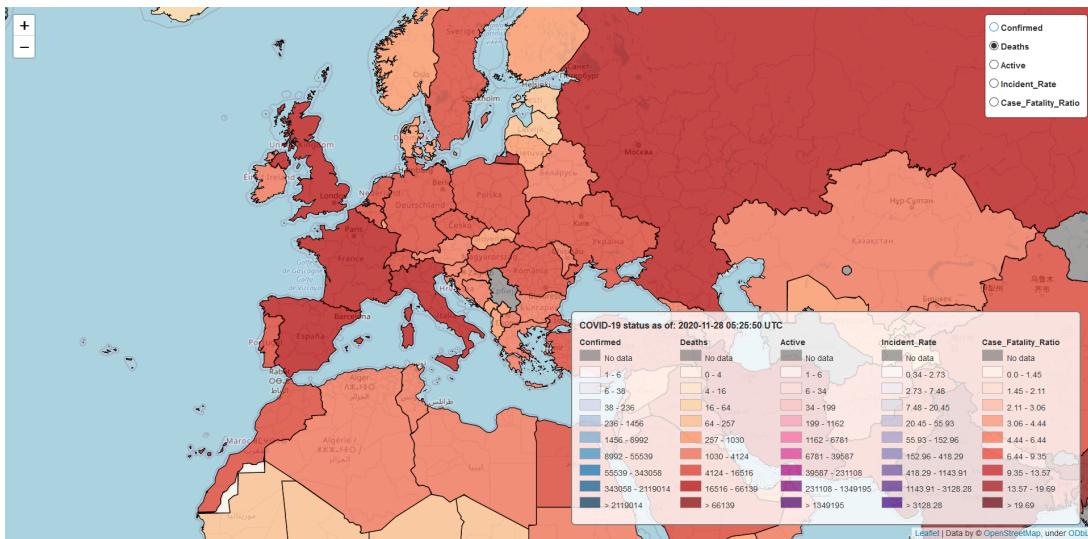
```

223     feature_group.add_to(map_covid)
224
225     # Aktywacja możliwości przełączania pomiędzy warstwami mapy
226     folium.LayerControl(collapsed=True).add_to(map_covid)
227
228     ...
229     Zapisanie ukończonej wizualizacji do odpowiedniego pliku
230     ...
231     map_covid.save(os.path.join(script_dir_path, '...', 'webapp', 'templates', 'COVID-19_viz.
232     ↪ html'))
232     print('Successfully created the COVID-19 viz!')

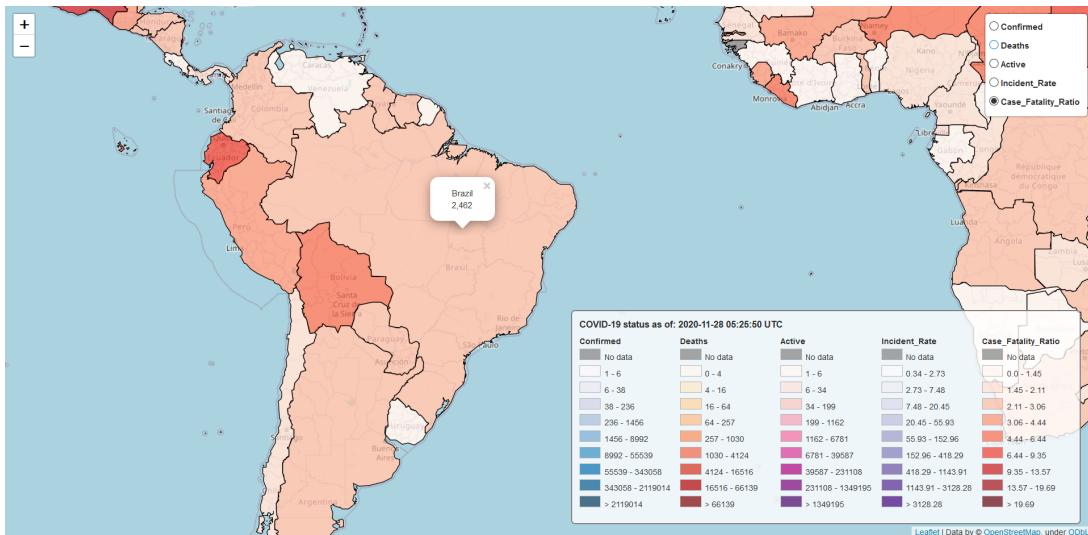
```



Rys. 4.4: Widok wizualizacji dla wybranej warstwy dot. wszystkich potwierdzonych przypadków koronawirusa



Rys. 4.5: Widok wizualizacji dla wybranej warstwy dot. wszystkich potwierdzonych zgonów spowodowanych koronawirusem



Rys. 4.6: Przykład działania wyskakującego okienka z informacją o dokładnej wartości z danej kategorii, w tym przypadku wartość współczynnika przypadków śmiertelnych dla Brazylii

4.2.6. Wizualizacja najnowszych przestępstw w San Francisco

Podprojekt dotyczący danych o przestępcości w San Francisco bazuje na wtyczce *MarkerCluster* do modułu *folium*. Klasa, opatrzona tą samą nazwą co wtyczka, posiada trzy kluczowe parametry wejściowe:

- *locations* - lista punktów zaznaczanych na mapie markerami,
- *popups* - lista okienek wyskakujących na skutek interakcji z markerami,
- *icons* - lista ikon przypisanych do poszczególnych markerów.

Parametr *locations*, to nic innego jak lista dwuelementowych list szerokości i długości geograficznej odpowiadająca miejscom, w którym doszło do złamania prawa. "Popup'y" zostały stworzone w taki sposób, żeby informować użytkownika o dokładnym znaku czasowym, dniu tygodnia i opisie danego zdarzenia. Do ikon przekazano tą samą grafikę dla każdego przestępstwa - pomarańczowy marker ostrzegawczy z białym wykrywnikiem.

Ponownie jak w kwestii wcześniejszej omówionych wizualizacji, wymagane było odrzucenie wszystkich rekordów bez wprowadzonej lokalizacji miejsca przestępstwa aby wtyczka działała w sposób poprawny. W tym przypadku było to szczególnie istotne - w zbiorze danych znalazło się całkiem sporo wpisów, gdzie dokładne miejsce złamania prawa nie było znane, co w naturalny sposób wynika ze specyfiki niektórych występków.

Wyniki działania kodu z listingu 4.6 przedstawiono na rysunkach 4.7 i 4.8.

Listing 4.6: Funkcja `create_sf_crime_viz` i jej zależności z pliku `sf_crime_viz.py`

```

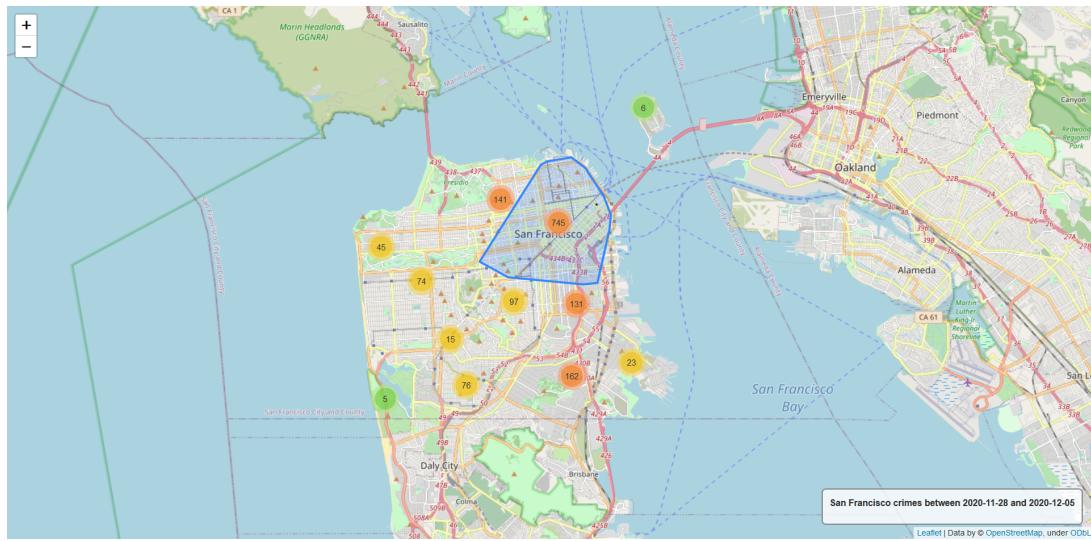
1 import os
2 import utils
3 import pandas as pd
4 import folium
5 from branca.element import Template, MacroElement
6 from folium.plugins import MarkerCluster
7 from datetime import datetime, timedelta
8
9 # Deklaracja istotnych ścieżek w strukturze projektu
10 script_dir_path = os.path.dirname(os.path.realpath(__file__))
11 data_dir_path = os.path.join(script_dir_path, '..', 'data')
12
13 def create_sf_crime_viz():
14     """
15         Załadowanie i wstępne przetworzenie danych o przestępcości w San Francisco
16

```

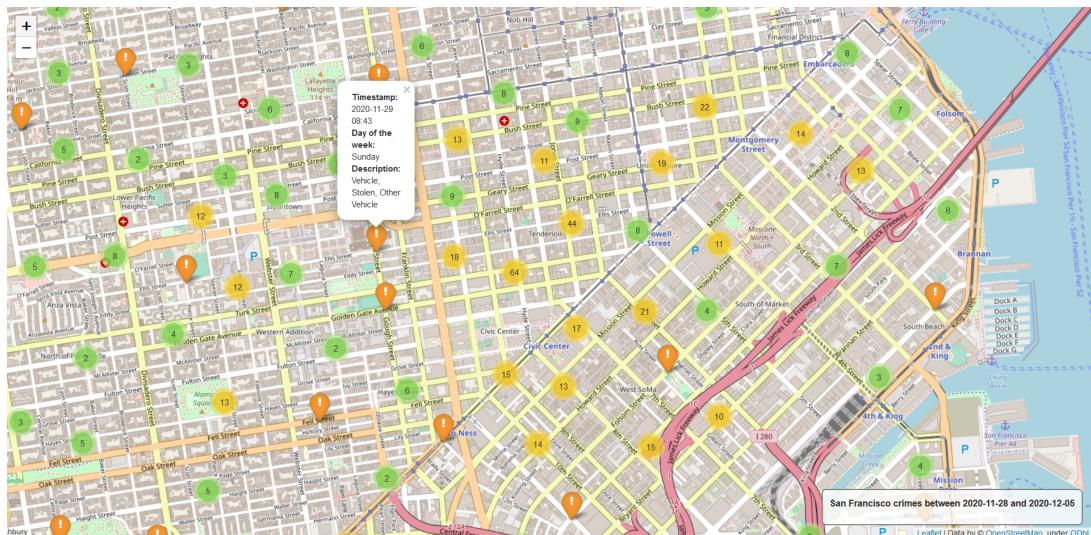
```

17 # Załadowanie danych z pliku do obiektu DataFrame
18 df_crime = pd.read_csv(os.path.join(data_dir_path, 'last_week_SF_crimes.csv'))
19
20 # Usunięcie rekordów, dla których nie wprowadzono współrzędnych geograficznych
21 df_crime = df_crime[df_crime['latitude'].notna()]
22 df_crime = df_crime[df_crime['longitude'].notna()]
23
24 ...
25 Inicjalizacja mapy
26 ...
27 map_crime = folium.Map(location=[37.773972, -122.431297],
28                         zoom_start=11,
29                         max_bounds=True,
30                         min_zoom=9,
31                         max_lat=38.5,
32                         max_lon=-122,
33                         min_lat=37,
34                         min_lon=-123)
35
36 ...
37 Stworzenie zawartości mapy
38 ...
39 # Stworzenie wyskakujących okienek (ang. popup) i ich zawartości
40 popups_list, locations_list = [], []
41 for _, row in df_crime.iterrows():
42     # Ucięcie niepotrzebnej części z informacji o znaku czasowym
43     incident_timestamp = row['incident_datetime']
44     incident_timestamp = incident_timestamp.replace('T', ' ')
45     incident_timestamp = incident_timestamp[:-7]
46
47     # Stworzenie obiektu typu popup i dodanie go do listy, która będzie konieczna do
        ↪ stworzenia klastra znaczników
48     popup_content = '<strong>Timestamp: </strong>' + incident_timestamp + '<br> \
49     + '<strong>Day of the week: </strong>' + row['incident_day_of_week'] +
        ↪ '<br> \
50     + '<strong>Description: </strong>' + row['incident_description']
51     popups_list.append(folium.Popup(html=popup_content))
52
53     # Dodanie długości i szerokości geograficznej do listy, która będzie konieczna do
        ↪ stworzenia klastra znaczników
54     locations_list.append(row[['latitude', 'longitude']].to_numpy().tolist())
55
56 # Stworzenie listy ikon znaczników ze spersonalizowaną grafiką
57 icon_list = []
58 for _ in range(len(locations_list)):
59     icon_list.append(folium.Icon(icon='exclamation', prefix='fa', color='orange'))
60
61 # Deklaracja klastra znaczników
62 marker_cluster = MarkerCluster(locations=locations_list, popups=popups_list, icons=
        ↪ icon_list)
63 marker_cluster.add_to(map_crime)
64
65 # Wyznaczenie przedziału czasowego, do którego należą wizualizowane dane
66 current_timestamp = datetime.now() - timedelta(days=1)
67 week_before = current_timestamp - timedelta(weeks=1)
68 current_timestamp = current_timestamp.strftime('%Y-%m-%d')
69 week_before = week_before.strftime('%Y-%m-%d')
70
71 # Stworzenie legendaria mapy (sam tytuł + przedział czasowy)
72 template = utils.create_legend(caption='San Francisco crimes between ' + week_before + '
        ↪ and ' + current_timestamp)
73 macro = MacroElement()
74 macro._template = Template(template)
75 map_crime.get_root().add_child(macro)
76
77 ...
78 Zapisanie ukończonej wizualizacji do odpowiedniego pliku
79 ...
80 map_crime.save(os.path.join(script_dir_path, '..', 'webapp', 'templates', 'SF_crime_viz.
        ↪ html'))
81 print('Successfully created the San Francisco crime viz!')

```



Rys. 4.7: Widok obszaru całego miasta San Francisco oraz poszczególnych zgrupowań znaczników. Na jechanie kursorem na symbol zgrupowania skutkuje wyświetleniem kształtu obszaru, który obejmuje



Rys. 4.8: Bardziej szczegółowy widok fragmentu miasta - zgrupowania automatycznie się dzielą i dostosowują do wielkości przybliżenia. Poziom zbliżenia widoczny na zrzucie ekranu umożliwia dostrzeżenie co niektórych pojedynczych znaczników, których wybranie skutkuje pojawieniem się widocznego "popup'u"

4.2.7. Wizualizacja wypadków drogowych w Wielkiej Brytanii (rok 2015)

Głównym narzędziem wykorzystanym w ostatnim podprojektie jest wtyczka *HeatMapWithTime* do modułu *folium*. Umożliwia ona implementację interaktywnej mapy termicznej zmieniającej się w czasie. W przeciwieństwie do wtyczki *TimeSliderChoropleth* nie dostarcza ona rozwiązań samego tylka suwaka i informacji o dacie lub godzinie, z którą związana jest aktualnie wyświetlana wizualizacja, ale również swego rodzaju odtwarzacz otwierający perspektywy tworzenia animacji, zapętlania ich czy też modyfikowania szybkości zmiany widoków [3].

Lista kluczowych parametrów wejściowych przyjmowanych przez konstruktor klasy *HeatMapWithTime*:

- *data* - lista lokalizacji zawierająca dane o szerokości i długości geograficznej oraz, opcjonalnie, o wadze danego rekordu,
- *index* - lista znaków czasowych, które znajdą się na osi czasu wizualizacji,
- *gradient* - parametr odpowiadający za personalizację wyglądu mapy termicznej, a konkretnie jej stylu kolorystycznego

Efekty działania programu z listingu 4.7 przedstawiono na rysunkach 4.9 i 4.10.

Listing 4.7: Funkcja create_uk_accidents_viz i jej zależności z pliku uk_accidents_viz.py

```

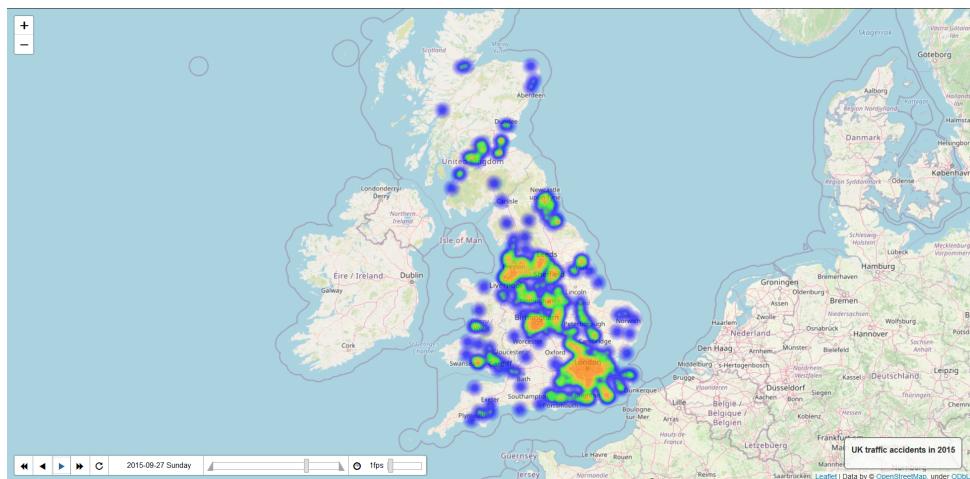
1 import folium
2 import os
3 import pandas as pd
4 import numpy as np
5 import utils
6 from folium.plugins import HeatMapWithTime
7 from branca.element import Template, MacroElement
8
9 script_dir_path = os.path.dirname(os.path.realpath(__file__))
10
11 def create_uk_accidents_viz():
12     """
13     Załadowanie i wstępne przetworzenie wizualizowanych danych
14     """
15     # Załadowanie danych o wypadkach drogowych do obiektu DataFrame
16     df_accidents_path = os.path.normpath(os.path.join(script_dir_path, '..', 'data',
17             'Accidents1115.csv'))
18     fields = ['Accident_Index', 'Latitude', 'Longitude', 'Date', 'Accident_Severity']
19     df_accidents = pd.read_csv(df_accidents_path, index_col='Accident_Index', usecols=fields)
20
21     # Sformatowanie i posortowanie danych po dacie
22     df_accidents['Date'] = pd.to_datetime(df_accidents['Date'], format='%Y-%m-%d', errors='raise')
23     df_accidents.sort_values('Date', inplace=True)
24
25     # Usunięcie rekordów, dla których nie wprowadzono współrzędnych geograficznych
26     df_accidents = df_accidents[df_accidents['Latitude'].notna()]
27     df_accidents = df_accidents[df_accidents['Longitude'].notna()]
28
29     # Pozostawienie danych dotyczących wyłącznie roku 2015, w celu ograniczenia wykorzystania
30     # pamięci
31     df_accidents = df_accidents[df_accidents['Date'].dt.year == 2015]
32
33     # Stworzenie obiektu przechowującego wszystkie daty - jeden z obowiązkowych parametrów wejściowych klasy HeatMapWithTime
34     heatmap_time_dates = df_accidents['Date'].dt.strftime('%Y-%m-%d %A').unique().tolist()
35
36     # Przypisanie wszystkim rekordom tej samej wagi
37     df_accidents['Weight'] = 0.8
38
39     # Stworzenie obiektu przechowującego wszystkie wypadki drogowe - kolejny z obowiązkowych
40     # parametrów wejściowych klasy HeatMapWithTime
41     heatmap_time_data = []
42     for date in heatmap_time_dates:
43         df_accidents_daily = df_accidents.loc[df_accidents['Date'] == date]
44         heatmap_time_data.append(df_accidents_daily[['Latitude', 'Longitude', 'Weight']].to_numpy().tolist())
45
46     """
47     Inicjalizacja mapy
48     """
49     map_accidents = folium.Map(location=[54, -2.4220],
50                             zoom_start=6,
51                             max_bounds=True,
52                             min_zoom=3,
53                             max_lat=60,
54                             max_lon=5,
55                             min_lat=49,
56                             min_lon=-12)
57
58     """
59     Stworzenie zawartości mapy
60     """
61     # Deklaracja obiektu na podstawie klasy HeatMapWithTime

```

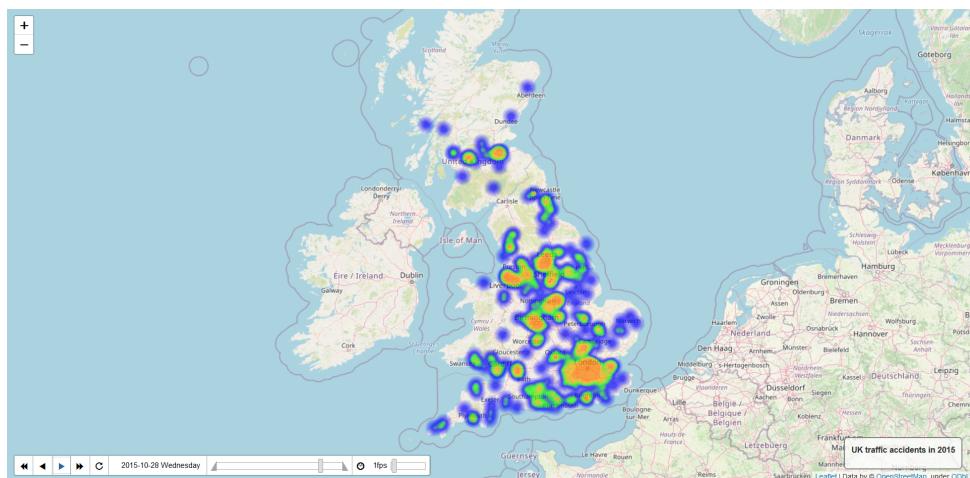
```

59     heatmap = HeatMapWithTime(heatmap_time_data,
60                             index=heatmap_time_dates,
61                             name='Traffic accidents in Great Britain (2015)',
62                             gradient={
63                               .8: 'blue',
64                               .95: 'lime',
65                               .998: 'orange',
66                               1: 'red'
67                             },
68                             use_local_extrema=False,
69                             min_opacity=0,
70                             max_opacity=0.7,
71                             scale_radius=False)
72     heatmap.add_to(map_accidents)
73
74 # Stworzenie legundy mapy (sam tytuł)
75 template = utils.create_legend(caption='UK traffic accidents in 2015')
76 macro = MacroElement()
77 macro._template = Template(template)
78 map_accidents.get_root().add_child(macro)
79
80 ...
81 Zapisanie ukończonej wizualizacji do odpowiedniego pliku
82 ...
83 map_accidents.save(os.path.join(script_dir_path, '..', 'webapp', 'templates', ' 
   ↪ UK_accidents_viz.html'))
84 print('Successfully created the UK accidents viz!')

```



Rys. 4.9: Mapa termiczna wypadków drogowych w Wielkiej Brytanii, 27 września 2015



Rys. 4.10: Mapa termiczna wypadków drogowych w Wielkiej Brytanii, 28 października 2015

4.3. Serwer webowy

4.3.1. Najważniejsze wykorzystane biblioteki

flask

Mikro-framework służący do budowania stron i serwisów internetowych. Cechą charakterystyczną narzędzia jest brak sprecyzowanych wymagań co do paczek i bibliotek, których należy używać równolegle z nim. Silną stroną *flaska* jest jego prostota oraz rozszerzalność, czyniąc go odpowiednim narzędziem do implementacji serwera na potrzeby omawianego projektu inżynierskiego [2].

flask_apscheduler

Rozszerzenie implementujące obsługę *Advanced Python Schedulera* w ekosystemie *flaska*. Dostarcza rozwiązań umożliwiających zlecanie cyklicznego wykonywania kodu w języku Python [16].

requests

Moduł stworzony z myślą o zapewnieniu eleganckiego i prostego sposobu na komunikację sieciową przez protokół HTTP z poziomu skryptów języka Python. Oddala od programisty konieczność np. własnoręcznego dodawania tekstowych zapytań do linków URL czy też kodowania danych metody POST [7].

sodapy

Pythonowe narzędzie klienckie do obsługi Socrata Open Data API. Zapewnia między innymi funkcjonalność wysyłania zapytań SoQL, w reakcji na które API odpowiada dostarczeniem wyłącznie tych danych, które spełniają warunek zapytania [8].

4.3.2. Funkcje aktualizacji danych

Na początku realizacji projektu stworzona została funkcja (listing 4.8), która służy jako ogólny szablon do pobierania plików z sieci. Ostatecznie została wykorzystana tylko w jednej wizualizacji, ponieważ źródło danych dla podprojektu dot. przestępcości w San Francisco dostarcza wsparcie dla wygodnego narzędzia SODA, aczkolwiek nie ulega wątpliwości, że istnienie tej funkcji jest dobrą praktyką programistyczną na przyszłość, uwzględniając potencjalny rozwój aplikacji o kolejne wizualizacje.

Listing 4.8: Funkcja download_file i jej zależności z pliku utils.py

```

1 import urllib
2
3 def download_file(download_url='', location='', filename=''):
4     if not download_url:
5         raise AttributeError('Error. No url provided')
6     if not filename:
7         filename = download_url[download_url.rfind('/')+1:]
8     if location:
9         location = os.path.join(location, filename)
10    else:
11        location = filename
12
13    try:
14        urllib.request.urlretrieve(download_url, location)
15    except Exception as e:
16        raise Exception(e)

```

Aktualizacja danych dot. COVID-19

Ponieważ dane, które stanowią podstawę do wizualizacji drugiego z podprojektów są cyklicznie publikowane na Githubie, należało napisać odpowiednią funkcję komunikującą się z jego API oraz dokonującą modyfikacji w lokalnym pliku.

Pierwszym etapem działania funkcji (listing 4.9) jest uzyskanie informacji o aktualności obecnie posiadanych danych. W momencie kiedy okaże się, że plik lokalny ma taką samą datę jak najnowszy plik w repozytorium, to działanie funkcji zostaje przerwane. Nazwę najnowszego dostępnego pliku określa się na podstawie odpowiedzi w formacie JSON, od API Githuba. Znajduje się on zawsze na przedostatniej pozycji listy plików w odpowiedzi, ostatnią lokatę zajmuje instrukcja *readme*.

W momencie kiedy nazwa pliku pomyślnie przejdzie weryfikację aktualności, rozpoczyna się właściwy proces pobierania przy użyciu wyżej opisanej funkcji *download_file* z modułu *utils*. W tej fazie dochodzi także do usunięcia przedawnionego pliku .csv (obsługa uwzględnia również możliwość obecności wielu przedawnionych plików).

Listing 4.9: Funkcja *download_covid_data* i jej zależności z pliku *covid_viz.py*

```

1 import os
2 import requests
3 import utils
4 import glob
5
6 data_dir_path = os.path.join(script_dir_path, '..', 'data')
7 try:
8     newest_dataset = glob.glob(os.path.join(data_dir_path, 'covid*'))[0].rsplit('/', 1)[-1]
9 except:
10    print('No Covid dataset found in the data folder')
11
12 def download_covid_data():
13     """
14         Pobieranie najnowszego dostępnego zbioru danych JHU CSSE COVID-19 z repozytorium kodu
15         ...
16         # Wywołanie metody GET w celu otrzymania informacji o najnowszym dostępnym pliku
17         folder_url = 'https://api.github.com/repos/CSSEGISandData/COVID-19/contents/
18             ↪ csse_covid_19_data/csse_covid_19_daily_reports'
19     try:
20         req = requests.get(folder_url)
21     except:
22         print('Could not get COVID-19 data from requested URL')
23         return
24
25     if req.status_code == 200:
26         req = req.json()
27     else:
28         print('Could not get COVID-19 data from requested URL')
29         return
30
31     global newest_dataset
32     newest_dataset_candidate = req[-2]['name']
33     print('Latest available dataset on Github: ' + newest_dataset_candidate)
34
35     # Sprawdzenie czy jest sens aktualizować dane
36     if newest_dataset.rsplit('_', 1)[-1] == newest_dataset_candidate:
37         print('Latest dataset is the most up to date one, aborting the viz update.')
38         return
39     else:
40         newest_dataset = newest_dataset_candidate
41
42     # Pobieranie najnowszego pliku z repozytorium i zastąpienie nim pliku lokalnego
43     Check if the newest available dataset on github is newer than the one currently possessed
44     download_url = 'https://raw.githubusercontent.com/CSSEGISandData/COVID-19/master/
45             ↪ csse_covid_19_data/csse_covid_19_daily_reports/' + newest_dataset
46     try:
47         utils.download_file(download_url=download_url, location=data_dir_path,
48                             filename='covid_' + newest_dataset)
49         covid_old_csv = glob.glob(os.path.join(data_dir_path, 'covid*'))
50         for filename in covid_old_csv:
51             if newest_dataset in filename:
52                 covid_old_csv.remove(filename)

```

```

51     for filename in covid_old_csv:
52         os.remove(filename)
53
54     except Exception as e:
55         print(str(e) + '\nCould not fetch newest dataset.')
56         return

```

Aktualizacja danych dot. przestępcości w San Francisco

Sposób komunikacji z API strony władz San Francisco wygląda zgoła inaczej niż w przypadku API portalu Github. Zastosowano tutaj wcześniej wspomiane SODA, którego rozwiązania poprawiają nieco efektywność programu aktualizującego. Stworzony został obiekt warunku *where*, który zostaje zaaplikowany do zapytania SoQL w momencie przekazania go jako parametru do metody *get* z klasy *Socrata* [8]. Olbrzymią zaletą tego rozwiązania jest brak potrzeby pobierania większego pliku zawierającego całość danych źródłowych (których nie jest mało, ponieważ są to wszystkie zgłoszenia o złamaniu prawa w San Francisco odnotowane od 2018 roku). Całość kodu funkcji przedstawiono na listingu 4.10.

Listing 4.10: Funkcja download_sf_crime_data i jej zależności z pliku sf_crime_viz.py

```

1 import os
2 import pandas as pd
3 from config import *
4 from sodapy import Socrata
5 from datetime import datetime, timedelta
6
7 data_dir_path = os.path.join(script_dir_path, '..', 'data')
8
9 def download_sf_crime_data():
10     """
11     Pobranie najnowszego zbioru danych dot. przestępcości w San Francisco
12     """
13     # Inicjalizacja klienta Socrata API
14     client = Socrata('data.sfgov.org', sf_data_token)
15
16     # Odjęcie jednego dnia od aktualnej daty, ponieważ aktualizacje danych zawsze wnoszą nowe
17     # informacje z dnia poprzedzającego dzień obecny
18     current_timestamp = datetime.now() - timedelta(days=1)
19     week_before = current_timestamp - timedelta(weeks=1)
20     current_timestamp = current_timestamp.strftime('%Y-%m-%d')
21     week_before = week_before.strftime('%Y-%m-%d')
22
23     # Deklaracja spersonalizowanego zapytania, które zostanie zaplikowane do metody GET
24     where_clause = 'incident_date between \'' + str(week_before) + 'T00:00:00.000\'' + ' and \'' +
25             str(current_timestamp) + 'T00:00:00.000\''
26
27     # Wykonanie metody GET
28     Getting the data with Socrata API and applying an SoQL clause to the downloaded .json
29     try:
30         # Oprócz klauzli where przekazane zostają również parametry informujące o
31         # identyfikatorze zbioru danych oraz o limicie rekordów otrzymywanych w
32         # odpowiedzi od API
33         results = client.get('wg3w-h783', where=where_clause, limit=100000)
34     except Exception as e:
35         print(str(e) + '\nCould not fetch newest dataset.')
36         return
37
38     df_results = pd.DataFrame.from_records(results)
39     df_results.to_csv(os.path.join(data_dir_path, 'last_week_SF_crimes.csv'))
40     print(df_results)

```

4.3.3. Zasadniczy kod serwera

Zadania

Plik *jobs.py* (listing 4.11) zawiera trzy funkcje, które wykorzystuje serwer. Pierwsza z nich, *covid_update*, to procedura aktualizacji danych oraz wygenerowania wizualiacji dot. rozwoju

pandemii COVID-19. Analogicznie działa metoda *sf_crime_update* dla wizualizacji przestępcości w San Francisco. Funkcja *create_all* uruchamiana jest podczas rozruchu serwera w celu zapewnienia obecności wszystkich wymaganych plików HTML-owych zawierających wizualizacje.

Listing 4.11: Plik jobs.py

```

1 import sys
2 import os
3
4 script_dir_path = os.path.dirname(os.path.realpath(__file__))
5 sys.path.append(os.path.join(script_dir_path, '..', 'viz'))
6 from covid_viz import download_covid_data, create_covid_viz
7 from sf_crime_viz import download_sf_crime_data, create_sf_crime_viz
8 from gdp_viz import create_gdp_viz
9 from uk_accidents_viz import create_uk_accidents_viz
10
11 def covid_update():
12     download_covid_data()
13     create_covid_viz()
14
15 def sf_crime_update():
16     download_sf_crime_data()
17     create_sf_crime_viz()
18
19 def create_all():
20     covid_update()
21     sf_crime_update()
22     create_gdp_viz()
23     create_uk_accidents_viz()

```

app.py

W pliku tym zawierają się wszystkie przypisania zawartości HTML-owych do odpowiednich ścieżek oraz inicjalizacja serwera. Podawanie dokładnej ścieżki do plików zawierających wizualizacje nie jest konieczne ze względu na funkcję *render_template*, które automatycznie szuka oczekiwanej zawartości w folderze *templates* [2]. Inicjalizacja serwera polega na początkowym uruchomieniu wszystkich jego zadań oraz dodaniu poleceń cyklicznych. Obie z zaleconych aktualizacji uruchamiają się w cyklu dobowym, o określonym czasie, który wyznaczony został na podstawie informacji o godzinie cyklicznego odświeżenia danych podanych na stronach właścicieli (dla danych covidowych jest to godzina 6:30 [17], dla danych o przestępcości w San Francisco 19:15). Zawartość pliku przedstawiono na listingu 4.12.

Listing 4.12: Plik app.py

```

1 from flask import Flask, render_template
2 from flask_apscheduler import APScheduler
3 import jobs
4
5 app = Flask(__name__)
6 scheduler = APScheduler()
7
8 @app.route('/')
9 def index():
10     return render_template('index.html')
11
12 @app.route('/covid-19-viz/')
13 def get_covid_viz():
14     return render_template('COVID-19_viz.html')
15
16 @app.route('/gdp-viz/')
17 def get_gdp_viz():
18     return render_template('GDP_viz.html')
19
20 @app.route('/sf-crime-viz/')
21 def get_crime_viz():
22     return render_template('SF_crime_viz.html')
23

```

```
24 @app.route('/uk-accidents-viz/')
25 def get_accidents_viz():
26     return render_template('UK_accidents_viz.html')
27
28 if __name__ == '__main__':
29     jobs.create_all()
30     scheduler.add_job(id='Covid-19 data update', func=jobs.covid_update, trigger='cron', hour
31         ↪ =6, minute=30)
32     scheduler.add_job(id='SF crime data update', func=jobs.sf_crime_update, trigger='cron',
33         ↪ hour=19, minute=15)
34     scheduler.start()
35     app.run()
```

Na rys. 4.11 przedstawiono komunikaty wyjściowe serwera pojawiające się podczas rozruchu aż do momentu osiągnięcia przez niego stanu pełnej funkcjonalności.

```
covid_12-05-2020.csv
Latest available dataset: 12-05-2020.csv
Latest COVID-19 dataset is the most up to date one, aborting the viz update.
Successfully created the COVID-19 viz!
Successfully created the San Francisco crime viz!
Successfully created the GDP viz!
Successfully created the UK accidents viz!
* Serving Flask app "app" (lazy loading)
* Environment: production
WARNING: This is a development server. Do not use it in a production deployment.
Use a production WSGI server instead.
* Debug mode: off
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Rys. 4.11: Informacje zwrotne z konsoli po uruchomieniu pliku serwerowego *app.py*

Rozdział 5

Podsumowanie

5.1. Osiągnięcia projektu

W ramach pracy inżynierskiej zrealizowano wszystkie główne założenia projektowe. Zredagowano odpowiednie wprowadzenie tematyczne, przeprowadzono przegląd i analizę wykorzystywanych danych, stworzono cztery kompleksowe podprojekty, które dotykają różnych istotnych zagadnień z obszaru wizualizacji danych oraz serwer wywiązuający się ze swoich, nieco bardziej podstawowych, zadań.

Clou projektu, czyli wizualizacje geograficzne, działają zgodnie z założeniami i w adekwatny oraz estetyczny sposób przedstawiają poszczególne dane, umożliwiając użytkownikowi różnego rodzaju interakcje, które mogą wspomóc go w procesie odnajdywania interesujących zależności.

Wizualizacja zmian w PKB per capita daje, bądź co bądź, uproszczony, ale stosowny wskaźnik rozwoju poszczególnych państw na przestrzeni lat. Prezentacja akutalnej sytuacji epidemicznej w sposób zbiorczy i szeroki dostarcza informacji o rozwoju pandemii koronawirusa uwzględniając różne kryteria jego oceny, spośród których użytkownik sam może wyciągnąć właściwe dla siebie wnioski. Mapa z oznaczeniami przypadków przestępcości w San Francisco może stanowić cenną informację dla mieszkańców miasta podczas podejmowania decyzji o osiedleniu się lub nieosiedleniu w danej dzielnicy. Wizualizacja termiczna zagęszczenia wypadków drogowych na obszarze Wielkiej Brytanii może być brana pod uwagę jako jedno z kryteriów w podjęciu decyzji o trasie jaką chce obrać kierowca wracający do domu w np. niedzielę lub w środę.

5.2. Kierunki dalszego rozwoju projektu

Najbardziej oczywistym kierunkiem potencjalnej rozbudowy projektu jest dodanie kolejnych podprojektów bazujących na innych danych, dotykających nowej tematyki i nowych technik wizualizacyjnych. Moduł *folium* zawiera jeszcze wiele narzędzi, których w tej pracy nie poruszono. Rozwiązania służące do rysowania tras, warstwy topograficzne, nakładki z plikami graficznymi, wykresy Vega - wszystko to aspekty znajdujące niezliczone zastosowania w dziedzinie wizualizacji, które mogą sprawić, że dany projekt będzie jeszcze bardziej interaktywny, czytelny oraz bogaty w informacje.

Warto w tym miejscu również wspomnieć o kilku funkcjonalnościach z wizualizacji już stworzonych, co do których możliwa jest jakaś forma rozbudowy. Przykładowo, dane dot. wypadków drogowych w Wielkiej Brytanii zawierają również informację o dotkliwości danego incydentu. Mogłyby to zostać wykorzystane poprzez nadanie poszczególnym rekordom wagi od niej zależnej i w efekcie zwiększyć nieco temperaturę mapy termicznej w obszarach, w których wypadki są na ogół bardziej niebezpieczne. Do rozważenia byłby także pomysł dodania do wizualizacji

drugiej mapy termicznej, przedstawiającej holistyczne, np. roczne zagęszczenie zdarzeń drogowych z podziałem na dni tygodnia.

Dane o przestępcości w San Francisco dotyczą bardzo obszernego zakresu informacji o konkretych wykroczeniach. Przykładem kolumny, która mogłaby się okazać wartościowym aspektem wizualizacji jest kategoria wykroczenia. Potencjalne rozwiązania filtrowania przypadków przestępstw lub różnorodnego "kolorowania" znaczników w zależności od powagi wykroczenia mogłyby wnieść dodatkowy wgląd w ocenę aktualnego stanu przestępcości w mieście.

Literatura

- [1] Dokumentacja menadżera paczek *conda*. <https://docs.conda.io/en/latest/>.
- [2] Dokumentacja modułu *flask*. <https://flask.palletsprojects.com/en/1.1.x/>.
- [3] Dokumentacja modułu *folium*. <https://python-visualization.github.io/folium/>.
- [4] Dokumentacja modułu *geopandas*. <https://geopandas.org/>.
- [5] Dokumentacja modułu *numpy*. <https://numpy.org/>.
- [6] Dokumentacja modułu *pandas*. <https://pandas.pydata.org/docs/>.
- [7] Dokumentacja modułu *requests*. <https://requests.readthedocs.io/en/master/>.
- [8] Dokumentacja moduły *sodapy*. <https://pypi.org/project/sodapy/>.
- [9] Dokumentacja systemu kontroli wersji *git*. <https://git-scm.com/doc>.
- [10] Dziesięć interaktywnych przykładów map i wizualizacji danych. *tableau.com*. <https://www.tableau.com/learn/articles/interactive-map-and-data-visualization-examples>.
- [11] Fragment oficjalnego dokumentacji języka Python dotyczący *pycache*. https://docs.python.org/3/library/sys.html?highlight=pycache#sys.pycache_prefix.
- [12] Kartografia - trochę historii. *gisplay.pl*. <https://gisplay.pl/kartografia/kartografia-historia.html>.
- [13] Portal data.worldbank, dane wykorzystane w wizualizacji PKB per capita [dostęp dnia 02.12.2020]. <https://data.worldbank.org/indicator/NY.GDP.PCAP.CD>.
- [14] Przykład kartogramu [dostęp dnia 02.12.2020]. <https://xdgov.github.io/data-design-standards/visualizations/choropleth-map>.
- [15] Repozytorium kodu modułu *branca*. <https://github.com/python-visualization/branca>.
- [16] Repozytorium kodu modułu *flask_apscheduler*. <https://github.com/viniciuschiele/flask-apscheduler>.
- [17] Repozytorium z wykorzystanymi danymi epidemicznymi [dostęp dnia 01.12.2020]. <https://github.com/CSSEGISandData/COVID-19>.
- [18] Segment strony data.gov.uk zawierający informacje o różnych zbiorach danych dot. wypadków drogowych [dostęp dnia 03.12.2020]. <https://data.gov.uk/dataset/cb7ae6f0-4be6-4935-9277-47e5ce24a11f/road-safety-data>.

-
- [19] Segment strony [data.sfgov.org](https://data.sfgov.org/Public-Safety/Police-Department-Incident-Reports-2018-to-Present/wg3w-h783) zawierający informacje o zbiorze danych dot. przestępcości w San Francisco (od roku 2018 do chwili obecnej) [dostęp dnia 03.12.2020]. <https://data.sfgov.org/Public-Safety/Police-Department-Incident-Reports-2018-to-Present/wg3w-h783>.
 - [20] Standard ISO 3166. <https://www.iso.org/iso-3166-country-codes.html>.
 - [21] Strona internetowa zawierająca ćwiczenia związane z wyznaczaniem szerokości i długości geograficznej [dostęp dnia 01.12.2020]. <http://www.jsu.edu/dept/geography/mhill/phygeogone/latlngprf.html>.
 - [22] Strona poświęcona systemom GIS [dostęp dnia 01.12.2020]. <https://gisgeography.com>.
 - [23] Strona wiki dot. formatu GeoJSON *git*. <https://en.wikipedia.org/wiki/GeoJSON>.
 - [24] Szablon ruchomej legendy mapy [dostęp dnia 29.12.2020]. <https://nbviewer.jupyter.org/gist/talbertc-usgs/18f8901fc98f109f2b71156cf3ac81cd>.
 - [25] Treść licencji Creative Commons Attribution 4.0. <https://datacatalog.worldbank.org/public-licenses#cc-by>.
 - [26] Treść licencji Open Data Commons Public Domain Dedication and License (PDDL) v1.0. <https://opendatacommons.org/licenses/pddl/1-0/>.
 - [27] Treść licencji Open Government Licence v3.0. <http://www.nationalarchives.gov.uk/doc/open-government-licence/version/3/>.
 - [28] Źródło zamieszczonego przykładu klastra znaczników [dostęp dnia 02.12.2020]. <https://mono.software/2017/12/29/google-maps-marker-clustering/>.
 - [29] Źródło zamieszczonego przykładu mapy termicznej [dostęp dnia 02.12.2020]. https://www.reddit.com/r/soccer/comments/2a6m2b/germany_vs_brazil_heat_map/.
 - [30] A. Cairo. *The Functional Art: An Introduction to Information Graphics and Visualization*. New Riders, wydanie 1, Stany Zjedoczone, 2012.
 - [31] L. G. Dong E, Du H. An interactive web-based dashboard to track COVID-19 in real time. Lancet Inf Dis., 2020. [https://www.thelancet.com/journals/laninf/article/PIIS1473-3099\(20\)30120-1/fulltext](https://www.thelancet.com/journals/laninf/article/PIIS1473-3099(20)30120-1/fulltext).
 - [32] J. Lawhead. *Learning Geospatial Analysis with Python*. Packt Publishing Ltd., wydanie 3, Wielka Brytania, 2019.
 - [33] E. Westra. *Python Geospatial Development*. Packt Publishing Ltd., wydanie 3, Wielka Brytania, 2016.