Raport z projektu: Analiza danych pogodowych

Zuzanna Dudek, Martyna Hałaj

Spis treści

1 Wprowadzenie

Celem niniejszego raportu jest przedstawienie projektu polegającego na analizie danych pogodowych, w szczególności temperatur, na podstawie ogólnodostępnych danych historycznych ze strony IMGW. Projekt obejmuje trzy główne etapy: pozyskanie i konsolidację danych, analizę pozyskanych danych wraz z wizualizacją wyników oraz wykrywanie anomalii pogodowych przy użyciu danych temperatury.

2 Przygotowanie danych

2.1 Pobieranie danych

Pierwszym zadaniem tej części jest pobranie oraz rozpakowanie 276 plików zip ze strony `IMGW`. Zostało to wykonane za pomocą kodu w `Pythonie` przy użyciu bibliotek `Pandas`, `os` oraz `Time`. Pierwsze linijki kodu definiują foldery docelowe, a kolejne tworzą je – jeden na pliki zip, a drugi na już rozpakowane pliki.

```
output_dir = "../pobrane"
output_dir_unzip = "../pobrane_unzip"

os.makedirs(output_dir, exist_ok=True)
os.makedirs(output_dir_unzip, exist_ok=True)
```

Następnie tworzone zostają dwie pętle (jedna w drugiej) iterujące lata i miesiące. Pętle odpowiednio zmieniają link, z którego pobierany jest plik. W pętli zawarte są także polecenia, które pobierają plik i go odpakowują.

```
for year in range(2001, 2024):
    for month in range(1, 13):
        filename = f"dane_{year}_{month:02d}.zip"
```

```
        url = f"https://danepubliczne.imgw.pl/data/
    dane_pomiarowo_obserwacyjne/dane_meteorologiczne/dobowe/klimat/{year}/{
    year}_{month:02d}_k.zip"

        print(f"Pobieranie: {url}")

        os.system(f'powershell Invoke-WebRequest -Uri "{url}"
        -OutFile "{output_dir}/{filename}"')
        os.system(f'powershell Expand-Archive -Path
        "{output_dir}/{filename}" -Force
        -DestinationPath {output_dir_unzip}"')
        time.sleep(1)
```

Kolejną komendą jest `time.sleep(1)`, dzięki której pobieranie wykonuje się co sekundę. Na końcu pętli następna pętla usuwa pliki, jeśli zaczynają się na „`k_d_t`“.

```
for file in os.listdir(f"{output_dir_unzip}"):
   if file.startswith('k_d_t'):
    os.remove(f"{output_dir_unzip}/{file}")
```

2.2 Scalenie danych

Scalanie wszystkich plików również zostało wykonane za pomocą kodu w `Pythonie`. Najpierw, jak poprzednio, zdefiniowane zostały foldery docelowe. Jeżeli folder nie istnieje, zostaje on utworzony.

```
output = r"../data/data.csv"
base_dir = r"../pobrane_unzip"

folder_path = os.path.dirname(output)
os.makedirs(folder_path, exist_ok=True)
```

Następna pętla w pierwszej iteracji tworzy docelowy plik, a następnie wybiera odpowiedni plik oraz folder, z którego sczytuje tabelę danych i dopisuje ją do pliku docelowego.

```
first = True
for year in range(2001, 2024):
   for month in range(1, 13):
    plik = f"k_d_{month:02d}_{year}.csv"
    folder = os.path.join(base_dir, plik)

    if os.path.exists(folder):
       df = pd.read_csv(folder, header=None,
       encoding='cp1250')

       df.to_csv(
          output,
          mode="w" if first else "a",
          index=False,
          header=False,
          encoding='utf-8'
       )
       first = False
```

2.3 Usunięcie niepotrzebnych kolumn

Ostatnim fragmentem tej części projektu jest usunięcie niepotrzebnych danych. Zostało to zrobione za pomocą biblioteki `Pandas`. Wczytana zostaje tabela z pliku, brane są jedynie potrzebne kolumny. Następnie nadpisany zostaje oryginalny plik.

```
df = pd.read_csv(output, header=None,
usecols=[1, 2, 3, 4, 5, 7, 9])

df.to_csv(
    output,
    mode="w",
    index=False,
    header=False,
)
```

3 Proste charakterystyki i statystyki danych

Druga część projektu skupia się na analizie oraz wizualizacji pobranych danych, a także wykrywaniu anomalii.

3.1 Sprawdzanie stacji z pełną historią

Pierwszym zadaniem w tej części jest sprawdzenie, ile stacji posiada pełną historię pomiarów. Zrobione jest to za pomocą komendy „`counts = df["Stacja"].value_counts()`“, która zlicza, ile razy dana nazwa stacji pojawia się w kodzie. Za pomocą pętli dla każdego miasta sprawdzamy, czy dana nazwa wystąpiła dokładnie 8400 razy (czyli liczba dni, z których są dane). Jeśli tak, to dane miasto zostaje dodane do listy z miastami o pełnej historii.

```
counts = df["Stacja"].value_counts()

full_hist=[]
for miasto in df["Stacja"]:
  if counts.loc[miasto] == 8400:
    if miasto not in full_hist:
      full_hist.append(miasto)
  else:
    continue
```

3.2 Prezentacja graficzna wybranej stacji

Do zadania drugiego wybrana została stacja `Przemyśl`, która miała jedynie 181 pomiarów. W celu zaprezentowania graficznego tych danych, najpierw stworzono nową tabelę, jedynie z danymi z wybranej stacji. Później dane pogrupowano według roku i miesiąca. Następnie dla każdej takiej grupy zliczono liczbę unikalnych dni, w których dostępne są pomiary. Piąta linia kodu zmienia otrzymane wyniki do postaci tabelarycznej.

```
unfull_city = df[df["Stacja"].isin(["PRZEMYŚL"])]

days_per_month = unfull_city.groupby(["Rok","Miesiąc"])
 ["Dzień"].nunique()
counts = days_per_month.unstack(fill_value=0)
```

Kolejnym krokiem było stworzenie pętli iterującej miesiące i lata, a w każdej iteracji były sprawdzane dwa warunki. Pierwszy z nich to sprawdzenie, czy dla każdej kombinacji miesięcy i lat w zbiorze danych występują

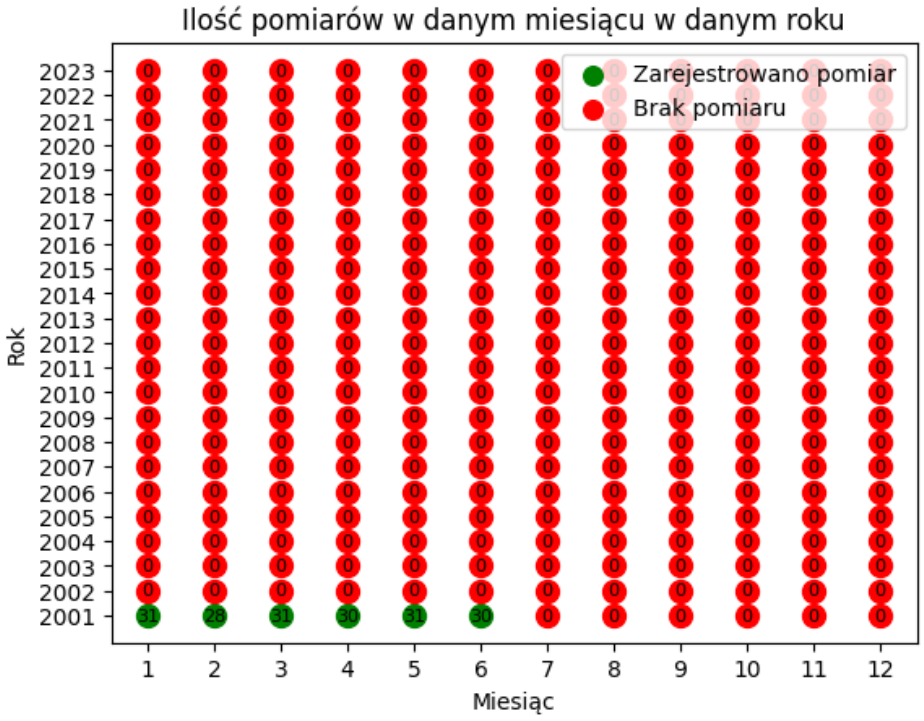
jakiekolwiek obserwacje z tego okresu. Jeżeli dane dla danego miesiąca i roku istnieją, to liczona jest liczba dni, która ma dane oraz tworzona jest zmienna `color=green`, która sprawia, że kropka na wykresie ma kolor zielony. Jeśli natomiast warunek nie jest spełniony, to `days=0`, a `color=red`. Na koniec tego warunku tworzony jest wykres kropkowy z parametrami miesięcy i lat wraz z odpowiednimi kolorami.

```
for y in range(2001,2024):
    for x in range(1,13):
        if ((unfull_city["Rok"] == y) & (unfull_city["Miesiąc"] == x)).any()
    :
                    days = counts.loc[y, x]
            color = "green"
        else:
            color = "red"
            days = 0
        plt.scatter(x, y, color=color, s=100)
```

Drugi warunek odpowiada za przypisanie odpowiednim kropkom (miesiącom w roku) liczby dni, w których zarejestrowany został pomiar, oraz wyświetlenie tego w odpowiednim miejscu na wykresie.

```
if days>=0:
    plt.text(x, y, str(days), color="black",
    ha='center', va='center', fontsize=8)
```

Ostatnia linijka kodu odpowiada za techniczne narysowanie wykresu, stworzenie legendy, tytułu i podpisanie osi.



Rysunek 1: Wykres przedstawiający liczbę pomiarów dla stacji Przemyśl

3.3 Wykres dziennej temperatury w zależności od minionych dni

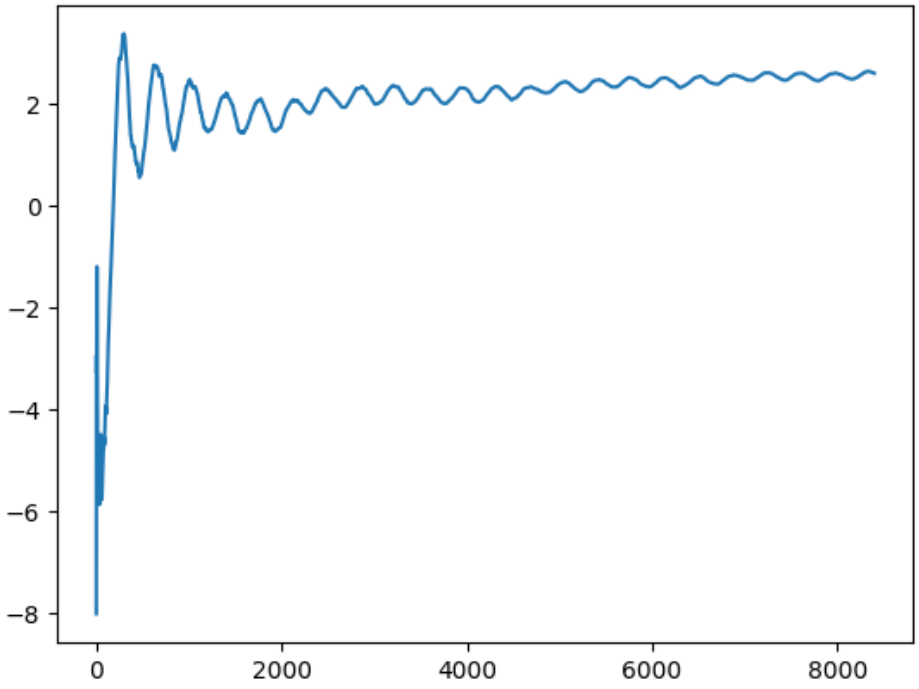
Polecenie trzecie rozpoczyna się od stworzenia kolejnej podtabeli dla wybranej stacji. Następnie stworzone zostają: pusta tabela na temperatury oraz zmienne liczące sumę temperatury oraz liczba minionych dni.

```
full_city = df[df["Stacja"].isin(["DOLINA PIĘCIU STAWÓW"])]

running_sum = 0
count = 0
running_avr_temp = []
```

Pętla przebiegająca po średnich temperaturach odpowiednio dodaje kolejne średnie temperatury do zmiennej `running sum` i zwiększa parametr `counts`, który jest liczbą dni liczonych do 1.01.2001. Następnie zmienna `running sum` zostaje podzielona przez `counts`, tworząc wynik po i–tym dniu. Na sam koniec iteracji otrzymany wynik zostaje dodany do tabeli temperatur.

```
for temp in full_city["Średnia temperatura"]:
    running_sum += temp
    count += 1
    current_mean = running_sum / count
    running_avr_temp.append(current_mean)
```



Rysunek 2: Wykres średniej dziennej temperatury w zależności od liczby dni, jakie upłynęły od 01.01.2001 dla stacji Dolina Pięciu Stawów

3.4 Szukanie najmniejszej i największej zarejestrowanej temperatury

Na początku tworzone są zmienne, do których przyporządkowane zostają kolumny. Do pierwszej zmiennej przyporządkowywana zostaje najniższa wartość wybranej kolumny, a do drugiej zmiennej - najwyższa.

```
lowest = df["Minimalna temperatura"].min()
highest = df["Maksymalna temperatura"].max()
```

Po tym stworzone zostały kolejne zmienne: `lowest_temp` oraz `highest_temp`. Zbiór danych został przefiltrowany w celu zlokalizowania konkretnych wierszy pasujących do tej wartości. Dane wyjściowe ograniczone są do nazwy stacji, temperatury i pełnej daty w celu dokładnejszej identyfikacji.

```
lowest_temp = df[df["Minimalna temperatura"] == lowest]
lowest_temp = lowest_temp[["Stacja", "Minimalna temperatura", "Dzień", "
    Miesiąc", "Rok"]]
highest_temp = df[df["Maksymalna temperatura"] == highest]
highest_temp = highest_temp[["Stacja", "Maksymalna temperatura", "Dzień",
    "Miesiąc", "Rok"]]
```

Najniższa temperatura została zarejestrowana dnia 8.01.2017 przez stację `JABŁONKA`.
Najwyższa temperatura została zarejestrowana dnia 8.08.2015 przez stację `CEBER`.

3.5 Wykres średniej temperatury miesięcznej dla każdego miesiąca każdego roku (podpunkt b)

Ta część zaczyna się od stworzenia nowej tabeli średnich temperatur miesięcznych. Dane ze stacji `Dolina Pięciu Stawów` pogrupowano względem roku oraz miesiąca, po czym obliczona została średnia wartość z kolumny „`Średnia temperatura`“ dla każdego miesiąca w badanym okresie. Wynik został przekształcony z powrotem do formatu tabeli.

```
full_city = df[df["Stacja"].isin(["DOLINA PIĘCIU STAWÓW"])]

avg_multi_group = (full_city.groupby(['Rok', 'Miesiąc'])['Średnia
    temperatura'].mean()).to_frame().reset_index()
```

Ze wszystkich wartości kolumny „`Średnia temperatura`“ obliczono wartości minimalną i maksymalną. Na tej podstawie ustalono skalę kolorów. W celach estetycznych numerom odpowiadającym miesiącom przypisano rzeczywiste nazwy.

```
norm = plt.Normalize(vmin=avg_multi_group['Średnia temperatura'].min(),
    vmax=avg_multi_group['Średnia temperatura'].max())
sm = plt.cm.ScalarMappable(cmap='coolwarm', norm=norm)

avg_multi_group['Miesiąc'] = avg_multi_group['Miesiąc'].apply(lambda x:
    calendar.month_name[x])
```

Dalej następuje pętla iterująca po wszystkich miesiącach we wcześniej utworzonej tabeli. W pętli utworzono nową tabelę, przefiltrowaną według konkretnego miesiąca. W następnym kroku przypisano odpowiedni kolor dla każdej wartości w tabeli. Następnie sporządzono wykres przy użyciu biblioteki `seaborn`. Linia 15 kodu tworzy legendę kolorów, która wizualizuje wartości temperatury.

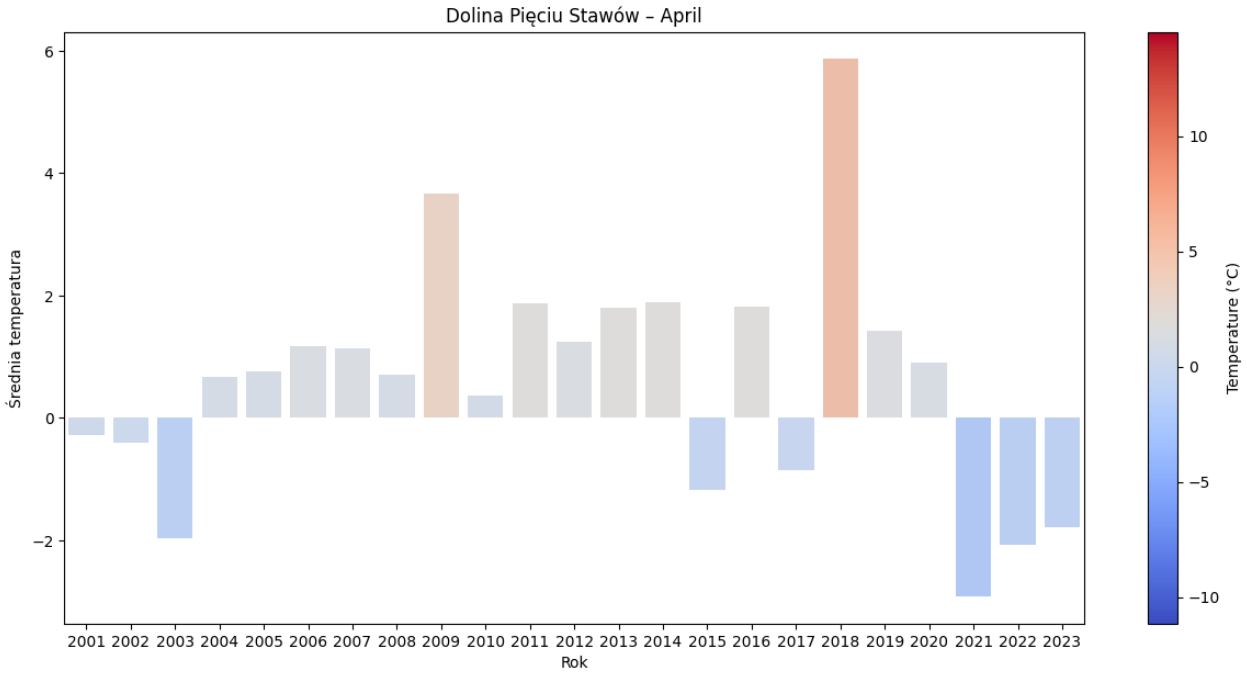
```
for month in avg_multi_group['Miesiąc'].unique():
    month_data = avg_multi_group[avg_multi_group['Miesiąc'] == month]
    fig, ax = plt.subplots(figsize=(15, 7))

    colors = [sm.to_rgba(val) for val in month_data['Średnia temperatura'
    ]]

```

```
    sns.barplot(data=month_data,
            x='Rok',
            y='Średnia temperatura',
            palette=colors,
            hue='Rok',
            legend=False
            )

    fig.colorbar(sm, ax=ax, label='Temperature')
    plt.title(f"Dolina Pięciu Stawów - {month}")
    plt.show()
```



Rysunek 3: Wykres średniej temperatury w Kwietniu dla stacji Dolina Pięciu Stawów

4 Wykrywanie anomalii

W ostatniej części projektu zadaniem było znalezienie sposobu na wykrycie anomalii pogodowych. Naszym sposobem było znalezienie najbardziej drastycznej różnicy pomiędzy najwyższą i najwyższą temperaturą w ciągu jednego dnia.
Kod odpowiadający za znalezienie takiej różnicy rozpoczęto od stworzenia nowej kolumny w tabeli głównej, poprzez odjęcie kolumny z najniższymi od kolumny z najwyższymi temperaturami. Następnie wyniki posegregowano od najwyższych do najniższych wartości kolumny `delta`.

```
df['delta'] = df['Maksymalna temperatura'] - df['Minimalna temperatura']

df[df['delta'] > 30].sort_values(by='delta', ascending=False)
```