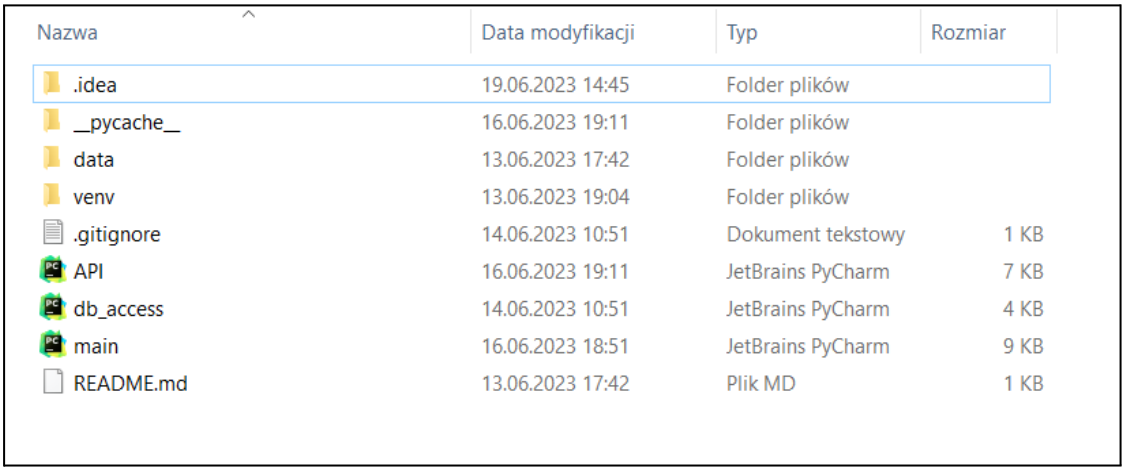
Opis kodu projektu

“Dziennik ciśnieniowca”

Marek Michelis, Jakub Pańtak, Szymon Pacyga

1. **Struktura projektu/repozytorium i wykorzystane biblioteki:**



Nazwa	Data modyfikacji	Typ	Rozmiar
.idea	19.06.2023 14:45	Folder plików	
__pycache__	16.06.2023 19:11	Folder plików	
data	13.06.2023 17:42	Folder plików	
venv	13.06.2023 19:04	Folder plików	
.gitignore	14.06.2023 10:51	Dokument tekstowy	1 KB
API	16.06.2023 19:11	JetBrains PyCharm	7 KB
db_access	14.06.2023 10:51	JetBrains PyCharm	4 KB
main	16.06.2023 18:51	JetBrains PyCharm	9 KB
README.md	13.06.2023 17:42	Plik MD	1 KB

Wykorzystane biblioteki:

- tkinter - utworzenie interfejsu graficznego programu,
- pandas - obsługa plików .csv i zaawansowane operacje na bazach danych z nich utworzonych,
- matplotlib - generowanie wykresów,
- tkcalendar - widget kalendarza do tkinter'a pozwalający wykorzystywany w naszym programie do ustawiania daty dla kolejno dodawanych wpisów,
- numpy - pozwalający na zaawansowane operacje matematyczne na danych, w naszym przypadku wykorzystywany przy tworzeniu podglądu listy wpisów.
-

2. **Kod main:**

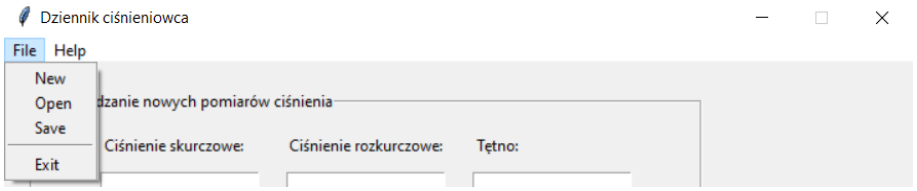
   W main znajduje się kod odpowiedzialny za wyświetlanie interfejsu, tworzenia wykresów oraz kilka funkcji.

   Utworzenie głównego okna programu oraz nadanie mu tytułu.

```
16  # okno
17  root = tk.Tk()
18  root.title("Dziennik ciśnieniowca")
19
```

Menu na pasku okna programu z przypisanymi funkcjami.

```python
# menu glowne
menubar = Menu(root)
filemenu = Menu(menubar, tearoff=0)
# new_file(show_frame)
filemenu.add_command(label="New", command=lambda: new_file(show_frame, show_frame_text))
filemenu.add_command(label="Open", command=lambda: select_file(show_frame, show_frame_text))
filemenu.add_command(label="Save", command=lambda: save_file())
filemenu.add_separator()
filemenu.add_command(label="Exit", command=root.quit)
menubar.add_cascade(label="File", menu=filemenu)
helpmenu = Menu(menubar, tearoff=0)
helpmenu.add_command(label="Help Index", command=donothing)
helpmenu.add_command(label="About...", command=donothing)
menubar.add_cascade(label="Help", menu=helpmenu)

```



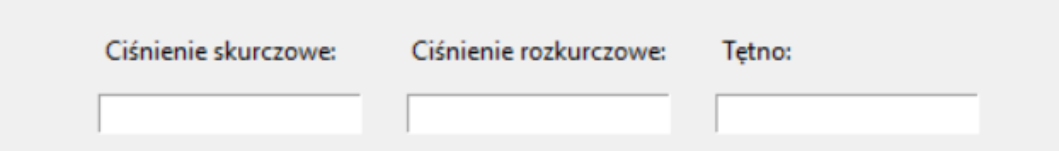
Podział głównego okna na frame'y tak aby każdy z elementów znajdował się w odpowiednim miejscu oraz umieszczenie ich w odpowiednich miejscach.

```python
# główny frame w oknie
main_frame = tk.Frame(root, padx=10, pady=10)
main_frame.pack()

frame_left = tk.Frame(main_frame, padx=10, pady=10)
frame_left.grid(row=0, column=0)

frame_right = tk.Frame(main_frame, padx=10, pady=10)
frame_right.grid(row=0, column=1)

```

```python
# frame do wprowadzania pomiarów
pressure_input_frame = tk.LabelFrame(frame_left, text="Wprowadzanie nowych pomiarów ciśnienia", padx=10, pady=10)
pressure_input_frame.grid(row=0, column=0)

# dodatkowe frame'y do oryganizacji szyku w pressure_input_frame
pif_frame_top = tk.Frame(pressure_input_frame, padx=10)
pif_frame_top.grid(row=0, column=0, columnspan=2)

pif_frame_bottom_left = tk.Frame(pressure_input_frame, padx=10, pady=10)
pif_frame_bottom_left.grid(row=1, column=0)

pif_frame_bottom_right = tk.Frame(pressure_input_frame, padx=10, pady=10)
pif_frame_bottom_right.grid(row=1, column=1)

```

Umieszczenie okien do wpisywania wartości ciśnień oraz opisy.

```
61    # ciśnienie skurczowe label i entry
62    label_systolic_pressure = tk.Label(pif_frame_top, text="Ciśnienie skurczowe:", anchor="w")
63    label_systolic_pressure.grid(row=0, column=0, sticky="w")
64    entry_systolic_pressure = tk.Entry(pif_frame_top)
65    entry_systolic_pressure.grid(row=1, column=0)
66
67    # ciśnienie rozkurczowe label i entry
68    label_diastolic_pressure = tk.Label(pif_frame_top, text="Ciśnienie rozkurczowe:", anchor="w")
69    label_diastolic_pressure.grid(row=0, column=1, sticky="w")
70    entry_diastolic_pressure = tk.Entry(pif_frame_top)
71    entry_diastolic_pressure.grid(row=1, column=1)
72
73    # tetno label i entry
74    label_heart_rate = tk.Label(pif_frame_top, text="Tętno:")
75    label_heart_rate.grid(row=0, column=2, sticky="w")
76    entry_heart_rate = tk.Entry(pif_frame_top)
77    entry_heart_rate.grid(row=1, column=2)
78
79    for widget in pif_frame_top.winfo_children():    # petla ustawiająca padx i pady dla
80        widget.grid_configure(padx=10, pady=5)       # wszystkich widgetów w danym frame
81
```

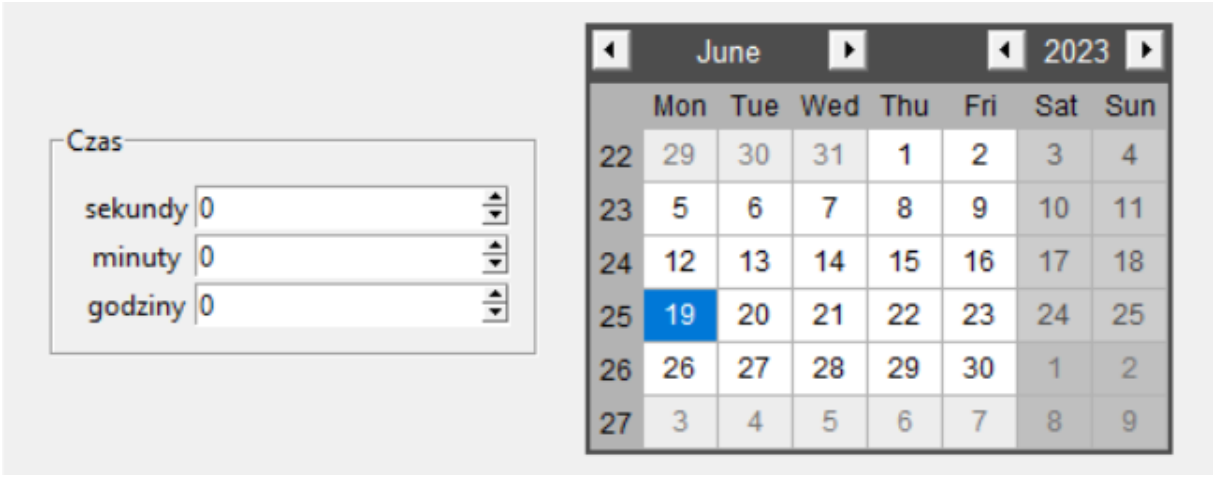


Dodanie kalendarza do wprowadzania daty pomiaru oraz umiejscowienie go.

```
90    # kalendarz do wprowadzania daty
91    cal = Calendar(pif_frame_bottom_right, selectmode='day', date_pattern='YYYY/mm/dd')
92    cal.pack()
93
```

Dodanie okienka do wprowadzania danych dotyczących czasu pomiaru ciśnienia.

```
# frame do wprowadzania czasu
time_input_frame = tk.LabelFrame(pif_frame_bottom_left, text="Czas", padx=10, pady=10)
time_input_frame.pack()

time_input_frame.columnconfigure(0, weight=1)
time_input_frame.columnconfigure(1, weight=1)
sec_input_frame = tk.Label(time_input_frame, text="sekundy")
sec_input_frame.grid(row=0, column=0,)
sec = tk.Spinbox(time_input_frame, from_=0, to=60)
sec.grid(row=0, column=1)
min_input_frame = tk.Label(time_input_frame, text="minuty")
min_input_frame.grid(row=1, column=0)
min = tk.Spinbox(time_input_frame, from_=0, to=60)
min.grid(row=1, column=1)

hour_input_frame = tk.Label(time_input_frame, text="godziny")
hour_input_frame.grid(row=2, column=0)
hour = tk.Spinbox(time_input_frame, from_=0, to=24)
hour.grid(row=2, column=1)

```



Dodanie przycisków do zapisu oraz usuwania danych wraz z ich funkcjami.

```
# przyciski do zapisu i usuwania
button_data_entry = tk.Button(pressure_input_frame, text="Zapisz pomiar",
                              command=lambda: gui_add_entry(f'{cal.get_date()} {hour.get()}:{min.get()}:{sec.get()}',
                                                            entry_systolic_pressure.get(),
                                                            entry_diastolic_pressure.get(),
                                                            entry_heart_rate.get(), show_frame_text, show_frame))
button_data_entry.grid(row=5, column=0, sticky="w"+"e", columnspan=3)

button_remove_last_data_entry = tk.Button(pressure_input_frame, text="Usuń wcześniej dodany pomiar",
                                          command=lambda: gui_delete_last_entry(show_frame_text))
button_remove_last_data_entry.grid(row=6, column=0, sticky="w"+"e", columnspan=3, pady=5)
```

Dodanie frame'ów do szukania pomiarów.

```
# frame do szukania pomiarów
search_measure_frame = tk.LabelFrame(frame_left, text="Wyszukaj pomiar ciśnienia", padx=10, pady=10)
search_measure_frame.grid(row=1, column=0, sticky="w"+"e")

smf_frame = tk.Frame(search_measure_frame)
smf_frame.pack(fill="x")

search_by_date_frame = tk.LabelFrame(smf_frame, text="Wyszukaj po dacie:")
search_by_date_frame.pack(fill="both", expand=True, side="left", padx=5, pady=5)

search_by_value_frame = tk.LabelFrame(smf_frame, text="Wyszukaj po wartości:")
search_by_value_frame.pack(fill="both", expand=True, side="right", padx=5,  pady=5)

```

Dodanie opisów, rozwijanego okna z możliwością wyboru ciśnienia oraz konfiguracja wszystkich tych elementów.

```

label_date_2 = tk.Label(search_by_date_frame, text="YYYY-mm-dd HH:MM:SS :")
label_date_2.pack()

entry_date_2 = tk.Entry(search_by_date_frame)
entry_date_2.pack()

for widget in search_by_date_frame.winfo_children():
    widget.pack_configure(padx=10, pady=5)

search_type_combobox = ttk.Combobox(search_by_value_frame, values=[" ", "Data", "Ciśnienie skurczowe",
                                                                   "Ciśnienie rozkurczowe", "Tętno"])
search_type_combobox.pack()

entry_type_value = tk.Entry(search_by_value_frame)
entry_type_value.pack()

for widget in search_by_value_frame.winfo_children():
    widget.pack_configure(padx=10, pady=5)

```

Funkcja, która rozpoznaje po jakim rodzaju ciśnienia lub czasie dokonać wyszukiwania.

```
1 usage
def change_variable():
    if len(entry_type_value.get()) != 0 and (search_type_combobox.get() == "Ciśnienie skurczowe" or
                                             search_type_combobox.get() == "Ciśnienie rozkurczowe" or
                                             search_type_combobox.get() == "Tętno"):
        search_variable = entry_type_value.get()
        search_db(show_frame, search_type_combobox.get(), search_variable, show_frame_text)
    elif len(entry_date_2.get()) != 0 and search_type_combobox.get() == "Data":
        search_variable = entry_date_2.get()
        search_db(show_frame, search_type_combobox.get(), search_variable, show_frame_text)

```

Dodanie przycisku wyszukiwania oraz umieszczenie go w gui.

```
180    button_data_entry = tk.Button(search_measure_frame, text="Szukaj", command=lambda: change_variable())
181    button_data_entry.pack(fill="x")
182
```

Opcje dotyczące wyświetlanego wykresu

```
183    # Plot options
184    plot_dp = IntVar()
185    plot_sp = IntVar()
186    plot_ht = IntVar()
187    plot_options = tk.LabelFrame(frame_left, text="Opcje wykresu", padx=10, pady=10)
188    plot_options.grid(row=2, column=0, sticky="w"+"e")
189    checkbox_show_sp = tk.Checkbutton(plot_options, text='Pokazuj Ciś. Sk.', variable=plot_sp, onvalue=1, offvalue=0)
190    checkbox_show_sp.pack(anchor="w")
191    checkbox_show_dp = tk.Checkbutton(plot_options, text='Pokazuj Ciś. Roz.', variable=plot_dp, onvalue=1, offvalue=0)
192    checkbox_show_dp.pack(anchor="w")
193    checkbox_show_ht = tk.Checkbutton(plot_options, text='Pokazuj Tętno', variable=plot_ht, onvalue=1, offvalue=0)
194    checkbox_show_ht.pack(anchor="w")
195    button_draw_plot = tk.Button(plot_options, text="Rysuj wykres", command=lambda: draw_plot(plot_sp.get(),
196                                                                                              plot_dp.get(),
197                                                                                              plot_ht.get()))
198    button_draw_plot.pack(fill="x")
199
```

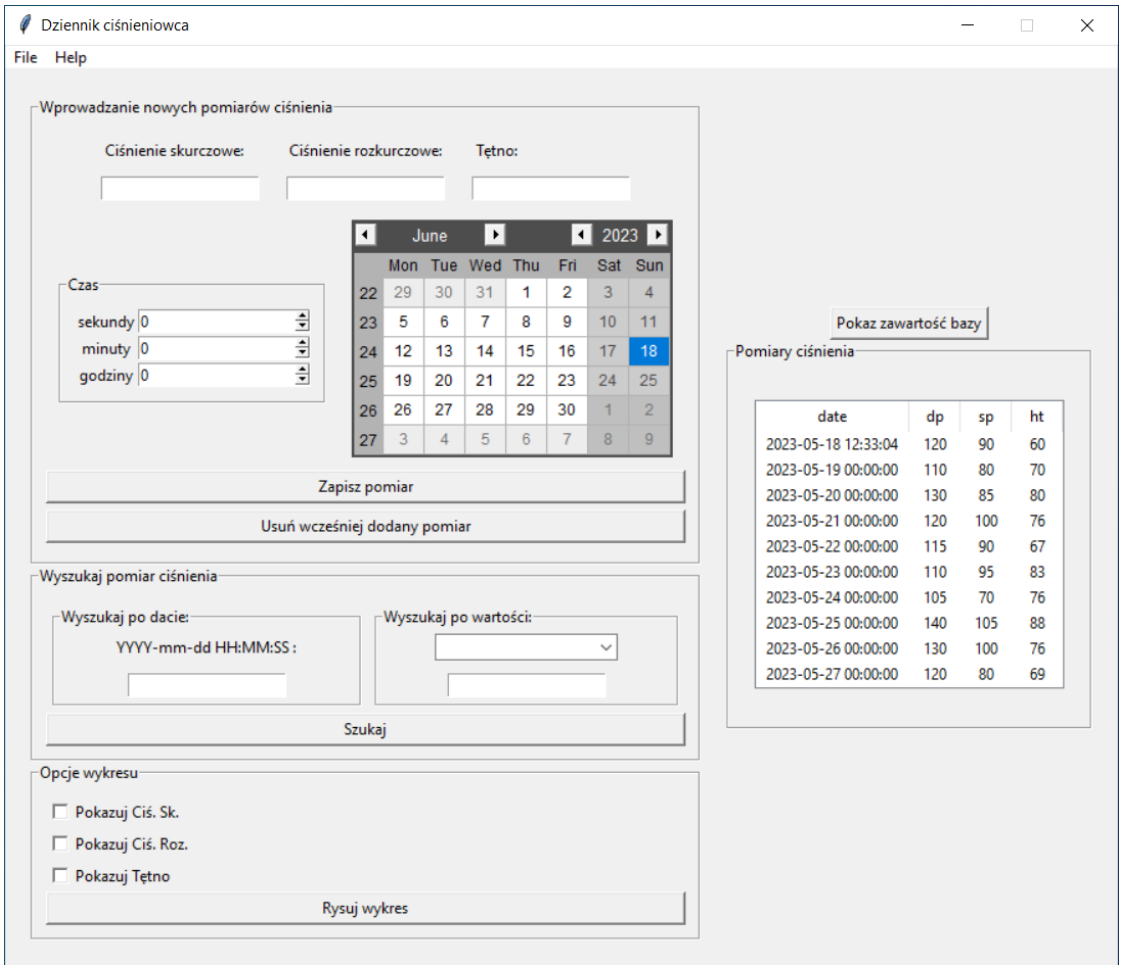
Przycisk do pokazywania zawartości aktualnie wczytanej bazy danych.

```
201    # pokazywanie wynikow w tym samym oknie
202    button_show_db = tk.Button(frame_right, text="Pokaz zawartość bazy", command=lambda: show_main_db(show_frame,
203                                                                                                      show_frame_text))
204    button_show_db.pack()
205    show_frame = tk.LabelFrame(frame_right, text="Pomiary ciśnienia", padx=10, pady=10)
206    show_frame.pack()
207    show_frame_text = tk.StringVar()
208    show_frame_text.set(str(show_db))
```

Opcja uniemożliwienia rozszerzania okna, dodanie menubar, automatyczne otwieranie się okna programu na środku ekranu oraz dodanie pętli pozwalającej na działanie gui.

```
212    root.resizable(False, False)
213    root.config(menu=menubar)
214
215    # otwieranie okna na środku ekranu
216    root.eval('tk::PlaceWindow . center')
217
218    #start GUI refresh loop
219    root.mainloop()
220
```

Końcowy efekt:



3. Kod db_access.py

“db_access.py” to plik zawierający klasę pozwalającą na obsłużenie funkcjonalności bazy danych w przygotowywanym programie.

```
df = pd.read_csv(os.path.join(os.getcwd(), "data/test_db.csv"))
```

Polecenie umożliwiające utworzenie testowego data frame’a pozwalającego na późniejsze przetestowanie niektórych funkcji programu.

Następnie mamy utworzoną właściwą klasę, która jest bardzo mocno wykorzystywana w kolejnych częściach programu.

```python
class Database:

    """A class to represent the dataframe..."""


    Marek Michelis +2
    def __init__(self, path=None):
        if path is not None:
            self.df = pd.read_csv(path)
            self.df["date"] = pd.to_datetime(df["date"], dayfirst=True, format='%d/%m/%Y %H:%M:%S')
        else:
            self.df = pd.DataFrame(columns=["date", "sp", "dp", "ht"])
```

Na początku konstruktor pozwalający na utworzenie obiektu “pustego” lub wypełnionego danymi z pliku o podanej ścieżce dostępowej.
W przypadku dołączenia pliku csv. program inicjalizuje obiekt właściwymi danymi jednocześnie ustawiając odpowiednie formatowanie kolumny odpowiedzialnej, za przechowywanie daty i czasu pomiarów. Przy inicjalizacji pustego obiektu tworzony jest data frame tylko i wyłącznie z nagłówkami kolumn.

```python
Marek Michelis
def __repr__(self) -> str:
    return self.df.to_string()
```

Zwrócenie dataframe’a w postaci stringa, bardzo przydatne podczas wyświetlania wyników.

```python
JPantak +1
def get_entry(self, n):
    return self.df.loc[n, :]
```

Metoda zwracająca n- pierwszych wyników z bazy danych

```python
Marek Michelis +2
def edit_entry(self, n, date: str = None, sp: int = None, dp: int = None, ht: int = None):
        if date is not None:
            self.df.at[n, "date"] = date
        if sp is not None:
            self.df.at[n, "sp"] = sp
        if dp is not None:
            self.df.at[n, "dp"] = dp
        if ht is not None:
            self.df.at[n, "ht"] = ht
```

Metoda pozwalająca na edycję n’tego wpisu w bazie danych. Możemy edytować jednocześnie edytować nieograniczoną ilość składowych danego pliku. W przypadku pozostawienia danego pola pustego, jego wartość w bazie nie jest zmieniana.

```python
1 usage (1 dynamic)   JPantak +1
def add_entry(self, date: str, sp: int, dp: int, ht: int) -> None:
    self.df.loc[len(self.df)] = [date, sp, dp, ht]
    self.df["date"] = pd.to_datetime(self.df["date"],dayfirst=True, format='%Y/%m/%d %H:%M:%S')
```

Metoda pozwalająca na dodanie wpisu do bazy danych, przyjmuje wszystkie niezbędne dane. Rekord jest umieszczany na końcu bazy danych. Pole daty jest od razu formatowane do ogólnie przyjętego w projekcie formatu.

```
JPantak +1
def sort_by_date(self, asc=True):
    self.df.sort_values(by="date", ascending=asc, inplace=True, ignore_index=True

JPantak +1
def sort_by_sp(self, asc=True):
    self.df.sort_values(by="sp", ascending=asc, inplace=True, ignore_index=True)

JPantak +1
def sort_by_dp(self, asc=True):
    self.df.sort_values(by="dp", ascending=asc, inplace=True, ignore_index=True)

JPantak +1
def sort_by_ht(self, asc=True):
    self.df.sort_values(by="ht", ascending=asc, inplace=True, ignore_index=True)
```

Metody pozwalające na po sortowanie bazy danych względem zaznaczonego parametru. Metoda wykonuje operacje bezpośrednio na utworzonej instancji pliku. Dodatkowo sortowanie zawsze jest w kolejności rosnącej.

```
1 usage (1 dynamic)   Marek Michelis
def save(self, path):
    self.df.to_csv(path, index=False)
```

Metoda pozwalająca na zapisanie zawartości aktualnie otwartej bazy danych do pliku we wskazanej lokalizacji.

```
Marek Michelis
def print(self):
    print(self.df)
```

Metoda pozwalająca na wypisanie w konsoli aktualnie obsługiwanej bazy danych.

```
1 usage (1 dynamic)   Marek Michelis
def del_entry(self):
    self.df.drop(len(self.df) - 1, inplace=True)
```

Metoda pozwalająca na usunięcie z bazy ostatniego dodanego wpisu.

```
4 usages (4 dynamic)  JPantak
def get_values(self, column: str):
    return self.df[column].values.tolist()
```

Metoda pozwalająca na pobranie z bazy danych kolumny o określonej nazwie. Wykorzystywane głównie przy tworzeniu wykresów.

```
4 usages (4 dynamic)  JPantak
def filter(self, date=None, sp=None, dp=None, ht=None):
    return self.df.loc[(self.df.date == date) | (self.df.sp == sp) | (self.df.dp == dp) | (self.df.ht == ht)]
```

Metoda pozwalająca na wyszukiwanie wpisów w bazie danych po konkretnym parametrze. Zwraca dataframe'a zawierającego wpisy o konkretnym polu.

```
1 usage  JPantak
def to_numpy(self):
    return self.df.to_numpy()
```

Metoda pozwalająca na zwrócenie dataframe'a w postaci gotowej do przetwarzania przy pomocy biblioteki numpy.

```
1 usage (1 dynamic)  Marek Michelis
def get_date(self):
    return self.df['date'].dt.strftime('%d.%.m%Y %H:%M:%S').values.tolist()
```

Metoda pozwalająca na zwrócenie listy stringów dat wpisów z bazy danych od razu sformatowanych w pożądany sposób. Wykorzystywane do ustawienia opisu osi poziomej wykresów.

4. **kod API.py**

Jest to plik zawierający implementacje najważniejszych funkcji programu. Dodatkowo pośredniczy pomiędzy interfejsem, a bazą danych.

```
# temp_date = ''
show_db = Database()
```

Utworzenie globalnego statycznego obiektu przechowującego bazą danych do aktualnego wyświetlenia dla użytkownika.

Funkcja pozwalająca na dodanie nowego wpisu do bazy danych. Dodatkowo zawiera opcję sprawdzenia poprawności wprowadzanych danych. Dodanie nowego wpisu automatycznie odświeża podgląd listy wpisów.

Funkcja implementująca funkcjonalność przycisku pozwalającego na usunięcie ostatniego wpisu z edytowanej bazy danych.

Funkcja pozwalająca na utworzenie nowej bazy danych. (Funkcja wywoływana z menu “File” >> New).

```
1 usage  Marek Michelis +1
def select_file(root,strv):
    global main_db
    global show_db
    filetypes = [
        ('csv files', '*.csv')
    ]
    path = filedialog.askopenfilename(filetypes=filetypes)
    main_db = Database(path)
    show_db = main_db
    refresh_trv(root, strv)
    strv.set(str(main_db))
```

Funkcja pozwalająca na za importowanie bazy danych z pliku na dysku. Po wczytaniu odpowiedniej listy rekordów, automatycznie odświeżany jest podgląd.

```
1 usage  Marek Michelis
def save_file():
    global main_db
    filetypes = [
        ('csv files', '*.csv')
    ]
    path = filedialog.asksaveasfile(filetypes=filetypes,defaultextension="*.*")
    main_db.save(path.name)

# draw plot
```

Funkcja pozwalająca zapisać aktualnie otwartą bazę danych do pliku pod wskazaną lokalizacją.

```
1 usage   Marek Michelis
def draw_plot(draw_sp, draw_dp, draw_ht):
    if draw_sp == 1 or draw_dp == 1 or draw_ht == 1:
        labels = []
        ylabel = []
        time_labels = []
        plot_x = main_db.get_values('date')
        plot_y3 = main_db.get_values('ht')
        plot_y2 = main_db.get_values('dp')
        plot_y1 = main_db.get_values('sp')
        plot_xticks = main_db.get_date()

        fig = plt.figure(layout='constrained')
        ax = fig.subplots()
        if draw_sp == 1:
            ax.plot(plot_x, plot_y1, color='tab:blue')
            labels.append('Ciś. Skurczowe')
        if draw_dp == 1:
            ax.plot(plot_x, plot_y2, color='tab:red')
            labels.append('Ciś. Rozkurczowe')

        if draw_dp == 1 or draw_sp == 1: ylabel.append('Ciśnienie [mmHg]')

        if draw_ht == 1:
            ax.plot(plot_x, plot_y3, color='tab:green')
            labels.append('Tętno')
            ylabel.append('Tętno [BPM]')

        ax.legend(labels, title='Zmienne')
        ax.set_ylabel(ylabel)
        ax.set_xlabel('Data')
        plt.xticks(plot_x, plot_xticks, rotation=45)
        plt.show()
```

Funkcja pozwalająca rysować wykresy o wskazanych parametrach. Jako parametry przyjmuje zmienne podłączone do checkboxów, za pomocą których możemy wybrać które informacje pojawią się na wykresie. Program generuje wykres w nowym oknie, automatycznie tworzy odpowiednią legendę, a także podpisuje osie. Warto zauważyć, iż opis osi Y jest zależny od wybranych parametrów wykresów. W celu lepszej analizy wykresu etykiety osi X są ustawiane jako daty w przyjaznym do odczytu formacie.

```
# search in db
2 usages  Marek Michelis +1
def search_db(root, type, variable, strv):
    global show_db
    print(type, variable)
    if type == "Data":
        show_db = main_db.filter(date=variable)
    elif type == "Ciśnienie skurczowe":
        show_db = main_db.filter(sp=int(variable))
    elif type == "Ciśnienie rozkurczowe":
        show_db = main_db.filter(dp=int(variable))
    elif type == "Tętno":
        show_db = main_db.filter(ht=int(variable))
    refresh_trv(root, strv)
    if strv is not None:
        strv.set(str(show_db))
```

Funkcja pozwalająca na wyszukiwanie w bazie danych wpisów o konkretnych parametrach. Za pomocą listy rozwijanej możemy wybrać, po wartości którego parametru chcemy filtrować naszą listę. Dodatkowo funkcja automatycznie wyświetla na liście podglądu nowo wygenerowaną listę wpisów (zgodną z zadanym parametrem wyszukiwania).

```
1 usage  Marek Michelis +1
def show_main_db(root, strv):
    global show_db
    show_db = main_db
    refresh_trv(root,strv)
    if strv is not None:
        strv.set(str(show_db))
```

Funkcja pozwalająca na wyświetlenie w oknie podglądu całej zawartości oryginalnej (nie przefiltrowanej) bazy danych.

```
5 usages   JPantak
def refresh_trv(root, show_frame_text):
    global show_db
    columns = ['date', 'dp', 'sp', 'ht']
    columns_width = [120, 40, 40, 40]
    trv = ttk.Treeview(root, selectmode='browse', height=10,
                       show='headings', columns=columns)
    trv.grid(row=4, column=4, columnspan=4, padx=10, pady=20)

    for i, col in enumerate(columns):
        trv.column(col, width=columns_width[i], anchor='c')
        trv.heading(col, text=str(col))
    for dt in show_db.to_numpy():
        v = [r for r in dt]
        trv.insert("", 'end', iid=v[0], values=v)
    show_frame = tk.LabelFrame(root, text="Pomiary ciśnienia", padx=10, pady=10)
    show_frame_text.set(str(show_db))
    left = tk.Label(show_frame, textvariable=show_frame_text)
    left.pack()
```

Funkcja obsługująca pole wyświetlające listę pomiarów. Tworzymy odpowiednią tabelę (pre definiujemy np. szerokości poszczególnych kolumn), a następnie dodajemy do niej kolejne wiersze zawierające nasze wpisy. Ten sposób prezentacji wyników pozwala na bardziej komfortowe użytkowanie programu (m. in. użytkownika ma możliwość przewijania listy).