
Wprowadzenie do analizy danych II

Wydanie 1.0

Dawid Michalik, Ewa Mardeusz

31 sty 2022

Contents:

1	Sprawozdanie	1
1.1	Zadanie 1	1
1.2	Zadanie 2	3
1.3	Zadanie 3	9
1.4	Podsumowanie	15

Ostrzeżenie: Uważać należy na odwołania w skryptach znajdując się ścieżki jak na przykład:

`/home/studentb10/Sprawozdanie_Dawid_Ewa/Zadanie_2/plik_a.csv`

Są to katalogi autorów dlatego prawdopodobnie należy dostosować je wcześniej do swoich potrzeb. Ponadto przez uruchomieniem skryptu należy zainstalować odpowiednie biblioteki w terminalu. W zadaniu 3 aby sprawdzić działanie uruchomić przez „main.py”

Linki do repozytoriów

1. Dawid Michalik - https://github.com/gadaki/Analiza_Projekt
2. Ewa Mardeusz - https://github.com/MardeuszEwa/Analiza_projekt

1.1 Zadanie 1

W pierwszym zadaniu należało opisać studium przypadku zastosowania języka python w analizie danych.

Dokument ten został utworzony jako strona HTML, w tym celu całość dokumentu zgodnie z poleceniem prowadzącego została spakowana oraz zapisana w repozytorium.

Poniżej znajdują się odnośniki do repozytoriów, każdego z autorów skąd można pobrać spakowany plik z rozszerzeniem .tar :

- Dawid Michalik: https://github.com/gadaki/Analiza_Projekt/tree/main/Zaliczenie
- Ewa Mardeusz: https://github.com/MardeuszEwa/Analiza_projekt/tree/main/Zaliczenie

Na początku utworzono wirtualne środowisko i zainstalowano pakiet komendami:

```
pipenv install
```

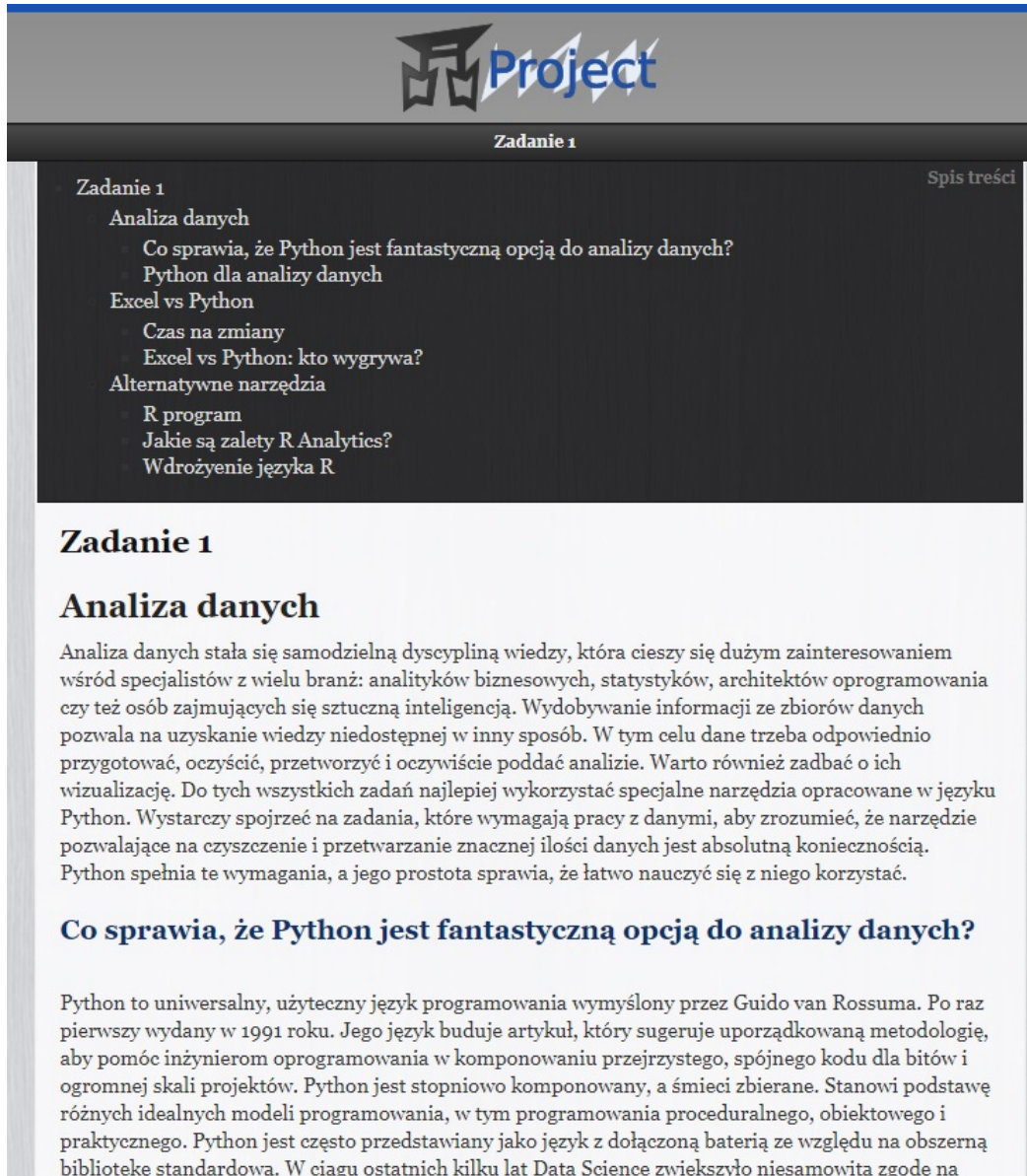
```
pipenv shell
```

```
pipenv install sphinx
```

ostatecznie aby uruchomić edytor wpisano komendę:

```
sphinx-quickstart
```

W ten sposób możliwe jest tworzenie plików tekstowych w pliku `index.rst`. Podczas tworzenia zmieniono domyślny motyw z „alabaster” na „pyramid” w pliku konfiguracyjnym. Wygląd motywu zaprezentowano na obrazku poniżej:



Rys. 1: Wygląd stronny HTML

Wszystko zapisano za pomocą komendy: `make html`. Ostatecznie spakowano pliki oraz wstawiono do repozytorium.

1.2 Zadanie 2

Zadanie numer dwa zostało podzielone na dwie oddzielne sekcje **a** oraz **b**.

Do wszystkich plików z rozszerzeniem .py dodano dodatkowo prace z rozszerzeniem .ipynb tak aby można było zobaczyć powstały wykres.

1.2.1 Zadanie 2a

W tym przypadku należało wygenerować zbiór danych zgodnych z wybranym przez siebie rozkładem prawdopodobieństwa.

W tym celu wygenerowano rozkład normalny o parametrach:

- Odchylenie standardowe: 10
- Liczebność: 70
- Środek: 0

Poleceniem poniżej przy użyciu pandas wygenerowano a następnie zapisano do pliku .CSV.

```
df = pd.DataFrame(np.random.normal(0,10,70))
df.to_csv('/home/studentb10/Sprawozdanie_Dawid_Ewa/Zadanie_2/plik_a.csv')
```

Otrzymujemy na przykład takie dane:

[4]:	Unnamed: 0	wartość
0	0	-23.680527
1	1	-8.083225
2	2	-10.543732
3	3	-2.950504
4	4	-5.793309
...
65	65	21.474103
66	66	6.102880
67	67	0.541958
68	68	-5.694902
69	69	4.406891

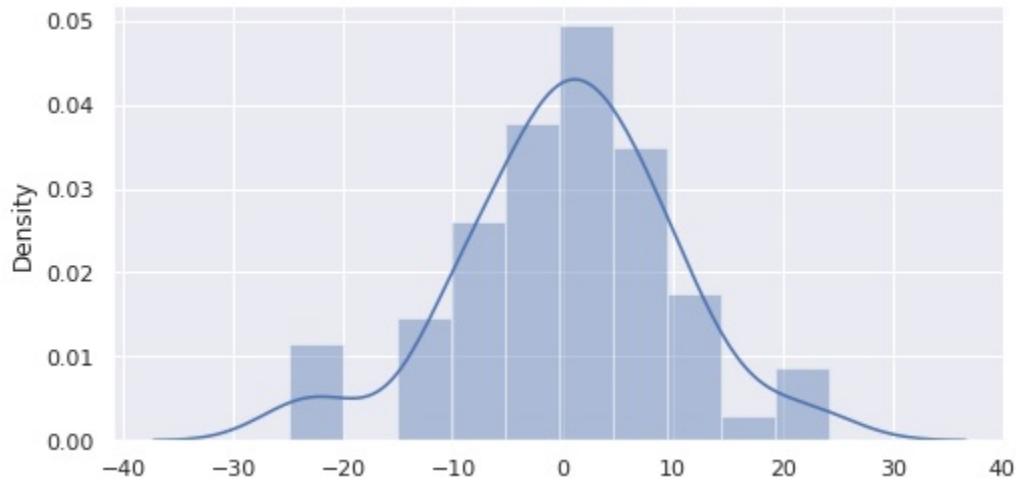
70 rows × 2 columns

Rys. 2: Przykładowe wartości rozkładu normalnego

Wybrano rozszerzenie .CSV ponieważ liczby generowane są liczbami rzeczywistymi dlatego owe rozszerzenie nie będzie miało problemu z odczytaniem/zapisem danych. Liczba wygenerowanych wartości nie jest duża - tylko 70 pozycji dlatego szybkość odczytu danych będzie satysfakcjonująca. Rozszerzenie to odczytuje czy dana liczba ma przed sobą „-” to znaczy czy jest minusowa. Odczytuje również pozycje po „,” czyli dziesiętne, tysięczne itp.

Ponadto uznano, iż format ten jest łatwo odczytywalny nawet w programach programowania obiektowego jak na przykład Excel.

Poniżej przedstawiono wygenerowane dane na wykresie.



Rys. 3: Wykres rozkładu normalnego

Wizualizacja potwierdza, iż jest to zgodnie z założeniami rozkład normalny lecz dla potwierdzenia wykonane zostały dwa testy:

- Test Shapiro-Wilk

```
from scipy.stats import shapiro
stats, p = shapiro(df)
print(stats, p)
if p > 0.05:
    print("Rozkład jest normalny")
else:
    print("Rozkład nie jest normalny")
```

- Test D'Agostino–Pearson

```
from scipy.stats import normaltest
stats, p = normaltest(df)
print(stats, p)
if p > 0.05:
    print("Rozkład jest normalny")
else:
    print("Rozkład nie jest normalny")
```

W ten sposób otrzymujemy wiadomość czy wartości napewno prezentują rozkład normalny.

Pamiętać należy aby przed odpaleniem skryptu nadać uprawnienia i użyć polecenia:

```
chmod u+x Zad_2.py
```

```
python Zad_2.py
```

Do prawidłowego działania warto pobrać biblioteki takie jak : numpy, seaborn czy pandas.

W środowisku maszynowym należy użyć: `pip install (nazwa)`

W środowisku wirtualnym `pipenv install (nazwa)`

Opisane wyżej czynności to schemat wykonywania poszczególnych kroków. Kompletnie wykonany skrypt w terminalu prezentuje się następująco:


```
[studentb10@calcnode Zadanie_2]$ python Zad_2a.py
Generowany jest zbiór danych o rozkładzie normalnym z parametrami, środek: 0 odchylenie standardowe: 10 liczebność: 70
Wygenerowany rozkład:
wartość
0 -12.350489
1 -1.288855
2 4.898345
3 -17.312590
4 21.956674
..
65 -0.950537
66 9.506724
67 1.232978
68 -4.513080
69 2.342477

[70 rows x 1 columns]
Poniżej następuje próba generacji wykresu lecz możliwość ta jest ograniczona i pokazywany jest tylko komunikat, który należy pominąć właściwy wykres znajduje się w sprawozdaniu.

/home/studentb10/.local/lib/python3.10/site-packages/seaborn/distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please use either `Figure.displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)

Sprawdzenie czy wygenerowany rozkład jest faktycznie rozkładem normalnym

Test D'Agostino-Pearson
[3.99848913] [0.13543756]
Wynik: Rozkład wygląda na normalny

Test Shapiro-Wilk
0.9831622838973999 0.46981726722717285
Wynik: Rozkład wygląda na normalny

Koniec Zadania 2a
[studentb10@calcnode Zadanie_2]$
```

Rys. 4: Zadanie_2a po wykonaniu w terminalu

1.2.2 Zadanie 2b

W kolejnym zadaniu należało przeprowadzić analize dowolnego szeregu czasowego.

Zostały wybrane średnie ceny ropy naftowej na giełdzie za rok 2021 w okresie od stycznia do grudnia.

Źródłem jest strona: <https://pl.investing.com/commodities/brent-oil-historical-data>.

Na wzór danych z danego okresu został sporządzony „słownik” z miesiącami i przypisanymi do nich wartościami. Kod został zaprezentowany poniżej.

```
dict = { 'Miesiąc': ['styczeń', 'luty', 'marzec', 'kwiecień', 'maj', 'czerwiec', 'lipiec',
                  'sierpień', 'wrzesień', 'październik', 'listopad', 'grudzień'], 'Cena' : [55.88, 66.13,
                  63.54, 67.25, 69.32, 75.13, 76.33, 72.99, 78.52, 84.38, 70.57, 77.78]
}
```

W kolejnym kroku za pomocą data frame zapisano do pliku z rozszerzeniem **.CSV** a następnie oczyszczono dane tak aby pozostałe same liczby do opisu danych.

Zapis do pliku **.CSV**:

```
df = pd.DataFrame(dict)
df.to_csv('/home/studentb10/Sprawozdanie_Dawid_Ewa/Zadanie_2/plik_b.csv')
```

	Miesiąc	Cena
0	styczeń	55.88
1	luty	66.13
2	marzec	63.54
3	kwiecień	67.25
4	maj	69.32
5	czerwiec	75.13
6	lipiec	76.33
7	sierpień	72.99
8	wrzesień	78.52
9	październik	84.38
10	listopad	70.57
11	grudzień	77.78

Wartość ropy naftowej w poszczególnych miesiącach można zobaczyć na powyższym zdjęciu

Pozostawienie samych wartości w postaci listy:

```
vals = (df['Cena'].tolist())
```

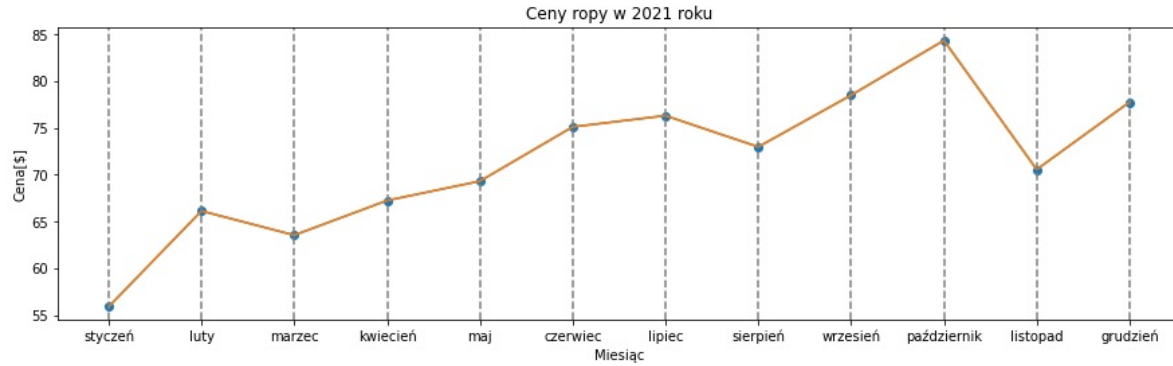
Kolejnym etapem zadania jest opis danych, w tym celu oczyszczone po zainstalowaniu odpowiednich bibliotek można poddać analizie wyliczone zostanie:

- Wartość maksymalna
- Wartość minimalna
- Wartość oczekiwana
- Mediana
- Kwartyl dolny
- Kwartyl górny
- Wariancja
- Odchylenie standardowe

```
Opisane dane:  
Wartość maksymalna: 84.38  
Wartość minimalna: 55.88  
Wartość oczekiwana: 71.485  
Mediana: 71.78  
Kwartyl dolny: 66.97  
Kwartyl górny: 76.6925  
Wariancja: 59.38173636363634  
Odchylenie standardowe: 7.705954604306746
```

Rys. 5: Opis danych obliczonych przez skrypt

Sporządzony został wykres cen ropy w 2021 roku który prezentuje się następująco:



Wykres cen ropy naftowej w roku 2021. Widać, iż sezonowość niewystępuje a trend jest możliwy lecz niepotwierdzony obliczeniowo. W tym celu zostanie zastosowane wygładzanie liniowe obliczona zostanie średnia kwartalna cen, jeżeli wartości będą wzrastać trend ten będzie wzrostowy.

Policzone zostały średnie kwartalne następującą metodą.

Zaimportowana została biblioteka *collections*.

```
n = 3 d = deque(maxlen=n) x = [d.append(e) or sum(d)/float(n) for e in vals][n-1:]
```

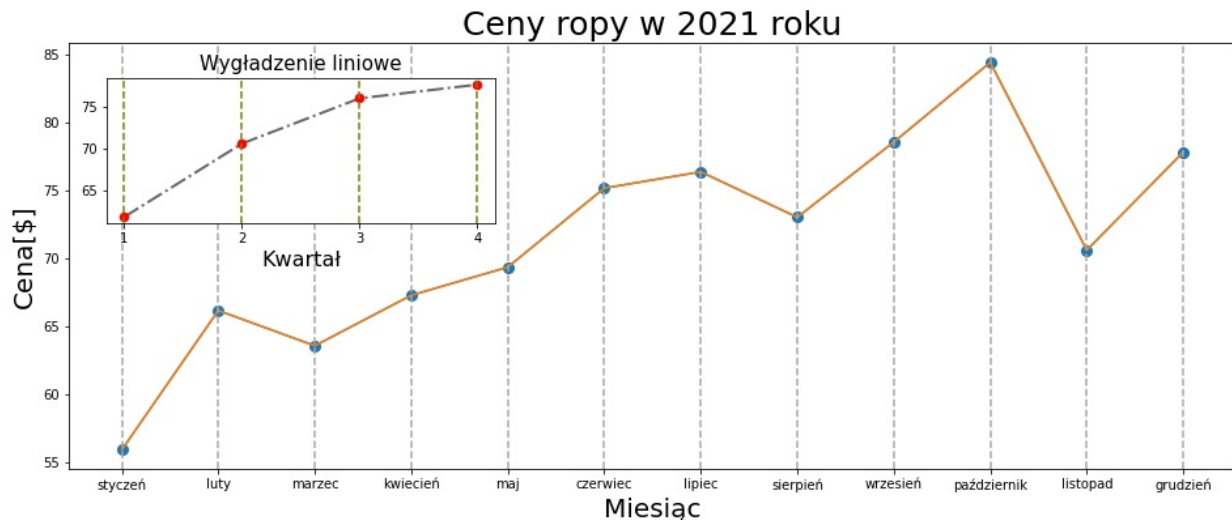
W ten sposób otrzymane zostały średnie z nich należy wydobyć wartości kwartalne.

```
x1 = [round(x[0],2),round(x[3],2),round(x[6],2),round(x[9],2)] print(x1)
```

Wyniki zostały zaokrąglone do 2 miejsc po przecinku a dane prezentują się następująco:

Średnia z poszczególnych kwartałów: [61.85, 70.57, 75.95, 77.58]

Ostatnim etapem jest porównanie zebranych danych w tym celu do sporządzonego wcześniej wykresu dodano wykres prezentujący średnią cen kwartalnych ropy za rok 2021.



Rys. 6: Ceny ropy naftowej ze średnimi kwartalnymi.

Na wygenerowanych wykresach widać, iż trend kwartalny jest wzrostowy ceny ropy naftowej w każdym z miesięcy wzrastały.

Wykresy zostały otrzymane za kodu przedstawionego poniżej:

```
fig, ax = plt.subplots(figsize=(16,6))
plt.xlabel('Miesiąc',size = 20)
plt.ylabel('Cena[$]', size = 20)
plt.title('Ceny ropy w 2021 roku',size = 25)
plt.plot(df["Cena"], marker='o',markersize = 8)
ax.plot(x, y)
for xc in df[('Miesiąc')]:
    plt.axvline(x=xc, color='darkgrey', linestyle='--')
ax2 = fig.add_axes([0.15, 0.560, 0.258, 0.258])
plt.xlabel('Kwartał',size = 16,)
plt.title('Wykładzenie liniowe',size = 15)
ax2.plot(values,x1,color = "dimgray", lw = 2, label = "Średnia",marker="o",
linestyle='dashdot',markeredgecolor = "red",markerfacecolor = "red",markersize = 6.5)
for xc2 in values:
    plt.axvline(x=xc2, color='olive', linestyle='--')
```

Po poprawnym uruchomieniu skryptu w tym przygotowaniu środowiska w terminalu otrzymamy następujące komunikaty.

```
(Zadanie 2) [studentb10@calcnode Zadanie_2]$ python Zad_2b.py
Zadanie_2b
Do utworzenia szeregu czasowego posłużono się średnimi cenami w miesiącu ropy naftowej na giełdzie za rok 2021
Dane są to ceny średnie ceny baryłki ropy za dany miesiąc dane odczytano ze strony: https://pl.investing.com/commodities/brent-oil-historical-data
Dane prezentują się następująco:
    Miesiąc  Cena
0   styczeń  55.88
1    luty   66.13
2   marzec  63.54
3   kwiecień 67.25
4    maj    69.32
5   czerwiec 75.13
6    lipiec  76.33
7   sierpień 72.99
8   wrzesień 78.52
9   październik 84.38
10  listopad 70.57
11  grudzień 77.78
Oczyszczona lista tylko z cenami: [55.88, 66.13, 63.54, 67.25, 69.32, 75.13, 76.33, 72.99, 78.52, 84.38, 70.57, 77.78]
Opisane dane:
Wartość maksymalna: 84.38
Wartość minimalna: 55.88
Wartość oczekiwana: 71.485
Mediana: 71.78
Kwartył dolny: 66.97
Kwartył górny: 76.6925
Wariancja: 59.38173636363634
Odchylenie standardowe: 7.705954604306746
Średnia z poszczególnych kwartałów: [61.85, 70.57, 75.95, 77.58]
Powyżej widać, iż trend jest wzrostowy ceny ropy w roku 2021 wzrastał z każdym miesiącem.
Koniec zadania 2b
(Zadanie 2) [studentb10@calcnode Zadanie_2]$
```

Rys. 7: Zadanie_2b w terminalu

Zadanie 2b pozwoliło na stwierdzenie, iż w roku 2021 ceny ropy naftowej wzrastały.

1.3 Zadanie 3

Przeprowadzono pomiary dotyczące punktualności odjazdów w komunikacji miejskiej. W ramach pomiarów zebrano od osób przeprowadzających pomiary następujące dane:

- 1.numer przystanku
- 2.numer linii
- 3.numer pojazdu
- 4.data
- 5.godzina planowa (z rozkładu)
- 6.godzina faktycznego przyjazdu
- 7.godzina faktycznego odjazdu

Dane zbierano zarówno przy użyciu aplikacji androidowej jak i ręcznie wpisując dane do tabeli i później wpisując je do arkusza kalkulacyjnego. Ponieważ pomiary przeprowadzane były przez wiele osób na różne sposoby następuje obawa o właściwy format danych. W efekcie końcowym pomiarów wszystkie zebrane dane dostępne są w formacie CSV.

W ten sposób wygenerowano 1000 wierszy danych, które prezentują się następująco:

1 ² ₃	Numer przystanku	1 ² ₃	Numer linii	1 ² ₃	Numer pojazdu	ABC 123	Data	ABC 123	Godzina planowana	ABC 123	Godzina faktycznego przyjazdu	ABC 123	Godzina faktycznego odjazdu
	30		142		532		25.07.2021 00:00:00		126-255		0,339583333		0,341666667
	5		102		654		14.03.2021 00:00:00		0,727083333		0,73125		0,731944444
	10		147		2		23.09.2021 00:00:00		0,140972222		0,14375		0,145833333
	15		8		6		01.08.2021 00:00:00		185-193		0,041666667		0,04375
	10		147		532		06.08.2021 00:00:00		0,349305556		0,348611111		0,349305556
	25		22		6532		19.11.2021 00:00:00		0,732638889		0,734722222		0,736805556
	25		10		948		08.12.2021 00:00:00		0,501388889		0,503472222		0,505555556
	5		102		654		08.03.2021 00:00:00		0,129861111		0,134027778		0,135416667
	40		147		42		12.12.2021 00:00:00		19.44		0,823611111		0,825694444
	35		145		332		10.06.2021 00:00:00		0,536805556		0,543055556		0,544444444
	52		10		948		20.06.2021 00:00:00		0,379861111		0,378472222	09'08	
	15		57		321		04.10.2021 00:00:00		0,141666667		0,140277778		0,142361111
	35		147		234		25.06.2021 00:00:00		0,098611111		0,102083333		0,104166667
	15		8		6		13.07.2021 00:00:00		0,109027778		0,109027778		0,110416667
	5		102		53		12.08.2021 00:00:00		0,133333333		01.03.2019 00:00:00	155:180	
	10		147		5		01.05.2021 00:00:00		0,634027778		0,634027778		0,636111111
	40		110		342		04.07.2021 00:00:00		0,538194444		0,544444444		0,545833333
	20		10		948	17,03,2021		0,6875		0,693055556		0,695138889	
	5		102		201		07.09.2021 00:00:00		0,772222222		0,777777778		0,778472222
	10		110		84		27.05.2021 00:00:00		0,303472222		01.07.2016 00:00:00		0,303472222
	35		243		332		13.01.2021 00:00:00		0,440972222	175:207			0,441666667
	10		110		84		10.12.2021 00:00:00		0,670833333		0,670833333		0,672222222
	30		102		42		15.02.2021 00:00:00		0,122916667		0,125		0,125694444
	20		10		948	29*12*2021		0,069444444		0,071527778		0,073611111	
	35		147		338		09.08.2021 00:00:00		0,754861111		0,755555556		0,756944444
	45		57		321		23.04.2021 00:00:00		0,222222222		0,226388889		0,228472222
	15		57		321		31.10.2021 00:00:00		0,125		0,131944444		0,134027778
	15		8		43		26.11.2021 00:00:00		0,164583333		0,165277778		0,167361111
	35		145		332		27.06.2021 00:00:00		0,057638889		0,0625	188:149	
	30		102		201		12.01.2021 00:00:00		0,565972222		0,570833333		0,572222222
	15		57		56		24.06.2021 00:00:00		0,359722222		0,363194444		0,364583333
	40		147		42		14.08.2021 00:00:00		0,456944444		0,455555556	01.10.1957 00:00:00	
	25		22		1		16.08.2021 00:00:00		0,161805556		0,164583333		0,166666667
	25		22		6532		19.12.2021 00:00:00		0,525694444		0,53125		0,533333333
	30		102		42		29.10.2021 00:00:00		0,0875		0,088888889		0,089583333
	45		95		94		24.06.2021 00:00:00		0,597222222		0,595833333	211:207	
	10		110		84	19,01,2021		0,197916667		0,204861111			0,206944444
	45		57		23		11.11.2021 00:00:00		0,721527778		0,720833333		0,722916667

Rys. 8: Zebrane dane do zadania 3

Już na prezentowanej tylko części danych widać, iż zawierają one błędy takie jak na przykład zły fromat daty, godziny odjazdu itp.

Do obróbki/poprawy podanych wyżej danych został stworzony pakiet „dataparser”.

Zostały stworzone odpowiednie pliki i dobranymi do nich rozszerzeniami takie jak:

- **setup.py** - pozwala zainstalować pakiet
- **__init__.py** - dzięki temu będzie możliwe używanie funkcji tego modułu w main.py
- **dataparser.py** - źródło kodu
- **main.py** - Dodatkowy skrypt pozwalający na łatwe uruchomienie kodu. (Pobrać razem z danymi z repozytorium)

Jako poprawny format daty został uznany - Dzień.Miesiąc.Rok

Jako poprawny format godzinowy został uznany format - „00:00”

Pakiet można pobrać i zainstalować od każdego z autorów kod https:

- Dawid Michalik - <https://github.com/gadaki/pakiet.git>
- Ewa Mardeusz - <https://github.com/MardeuszEwa/PakietAnaliza.git>

Aby zainstalować pakiet bezpośrednio z repozytorium należy użyć funkcji:

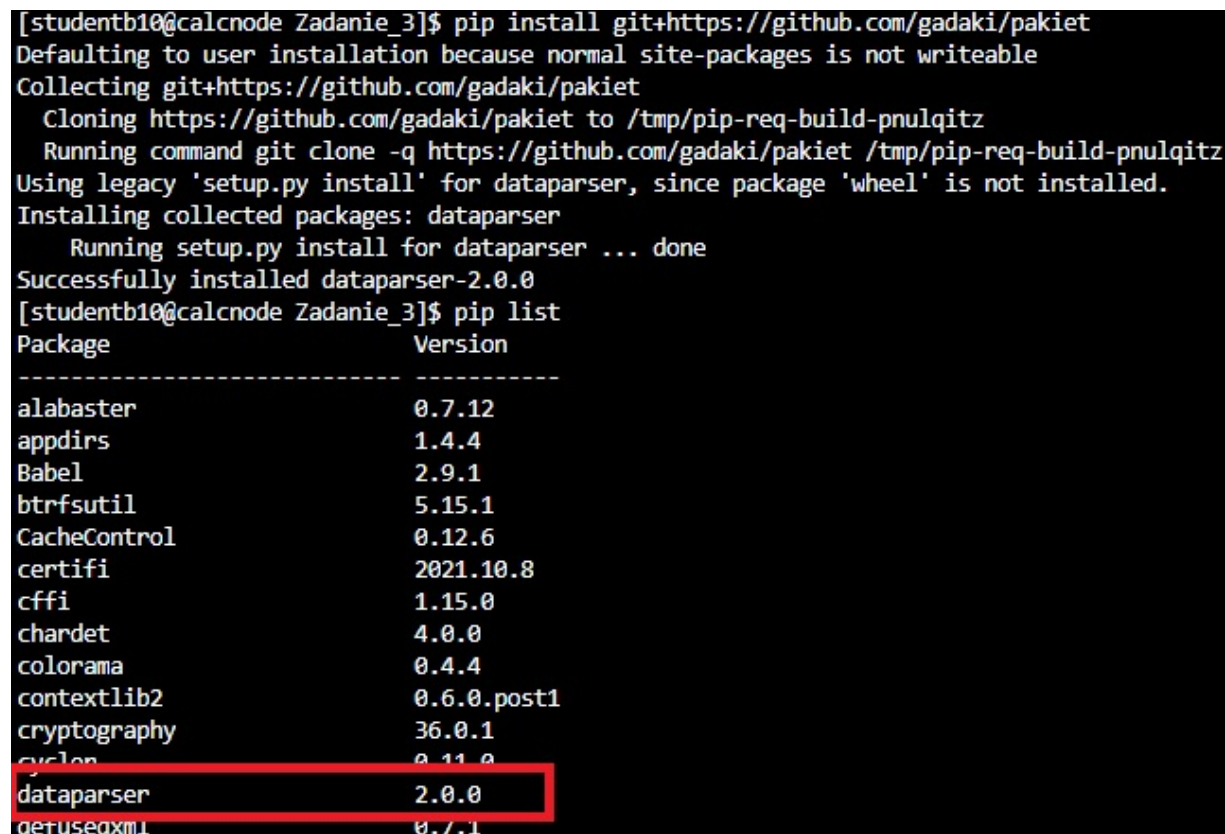
Środowisko wirtualne:

```
pipenv install git+https://github.com/gadaki/pakiet#egg=dataparser
```

Środowisko maszynowe

```
pip install git+https://github.com/gadaki/pakiet
```

Kroki te zainstalują pakiet i w terminalu otrzymamy następujące komunikaty:



```
[studentb10@calcnode Zadanie_3]$ pip install git+https://github.com/gadaki/pakiet
Defaulting to user installation because normal site-packages is not writeable
Collecting git+https://github.com/gadaki/pakiet
  Cloning https://github.com/gadaki/pakiet to /tmp/pip-req-build-pnulgitz
  Running command git clone -q https://github.com/gadaki/pakiet /tmp/pip-req-build-pnulgitz
Using legacy 'setup.py install' for dataparser, since package 'wheel' is not installed.
Installing collected packages: dataparser
  Running setup.py install for dataparser ... done
Successfully installed dataparser-2.0.0
[studentb10@calcnode Zadanie_3]$ pip list
Package            Version
-----
alabaster           0.7.12
appdirs             1.4.4
babel               2.9.1
btrfsutil           5.15.1
CacheControl        0.12.6
certifi             2021.10.8
cffi                1.15.0
chardet             4.0.0
colorama            0.4.4
contextlib2         0.6.0.post1
cryptography        36.0.1
cycler              0.11.0
dataparser          2.0.0
getuserxml          0.7.1
```

Rys. 9: Terminal po zainstalowaniu pakietu

Wywołanie listy potwierdza prawidłowe zainstalowanie pakietu.

Ze względu na długość kodu wybrane zostały najważniejsze elementy, które zostały opisane. Szczegółowe komentarze zostały zamieszczone bezpośrednio w skryptach.

Pierwszym krokiem tworzenia było stworzenie przystanków, przypisanych do nich numerów lini oraz pojazdów fragment kody znajduje się poniżej:

```
_valid_stops = {
    5: {
        "lines": {
            102: {"vehicles": [201, 42, 53, 654, 75, 12]}
```

Kolejnym etapem jest sprawdzenie czy dany plik występuje następnie odczyt jego i zapisanie w postaci listy. Na tym etapie korygowane już są możliwe pierwsze błędy usunnie „spacje”.

Następnie prowadzona jest korekcja, próba naprawy danych. Tworzona jest lista dla poprawnych oraz niepoprawnych danych.

Wprowadzona pętla sprawdza każdy wiersz pod kątem:

- Sprawdzanie długości listy czy posiada odpowiednią ilość kolumn

```
if len(split_line) != 7:
    items_nok.append(line)
    continue
```

- Sprawdzenie czy wartości są liczbami

```
if not split_line[0].isnumeric() or not split_line[1].isnumeric() or not
split_line[2].isnumeric():
    items_nok.append(line)
    continue
```

- Zamiana typu str na int

- Sprawdzenie numeru przystanku

```
if stop_number not in _valid_stops:
    items_nok.append(line)
    continue
```

- Sprawdzenie numeru lini

```
if line_number not in _valid_stops[stop_number]["lines"]:
    items_nok.append(line)
    continue
```

- Sprawdzenie numeru pojazdu

```
if vehicle_number not in _valid_stops[stop_number]["lines"][line_number]["vehicles"]:
    items_nok.append(line)
    continue
```

- Sprawdzenie i próba naprawy separatora daty
- Próba naprawy separatora godziny faktycznego przyjazdu
- Próba naprawy separatora godziny faktycznego odjazdu

- Zapis poprawnych danych w formacie Dictionary

Po wstępnej selekcji i naprawach obliczone jest opóźnienie w sekundach. kod został zaprezentowany poniżej.

```
def calc_delay_and_layover_time(data):  
    for item in data:  
        delay = item["real_time"] - item["time"]  
        item["delay"] = int(delay.total_seconds())
```

Czas przebywania na przystanku:

```
layover = item["real_departure_time"] - item["real_time"]
```

Po przeprowadzonych operacjach następuje utworzenie tabeli w formacie sqlite tworzone są tabelki z nazwami. Następnie prowadzona jest konwersja danych z typu datetime na string w danym formacie. Schemat zaprezentowano poniżej.

```
date = item["date"].strftime("%d.%m.%Y")  
time = item["time"].strftime("%H:%M")  
real_time = item["real_time"].strftime("%H:%M")  
real_departure_time = item["real_departure_time"].strftime("%H:%M")
```

Konwersja z sekund na docelowy format przedstawiony na początku.

```
delay = _seconds_to_time(item["delay"]) layover = _seconds_to_time(item["layover"])
```

Na końcu należy wszystko dodać do wcześniej utworzonej bazy danych.

Ostatnim etapem jest przetworzenie danych z sekund na format „00:00”.

Zdecydowano się na zapis informacji o ujemnej ilości sekund i odrócenie znaku. Ma to na celu ułatwić dalszą konwersję.

```
negative = False  
if seconds < 0:  
    negative = True  
    seconds *= -1
```

Następnie obcięto część dziesiętną, oraz zapisano minuty oraz sekundy. Dodano również znak „-” gdy autobus przyjeżdżał zbyt wcześnie.

```
m = seconds // 60  
s = seconds - (m * 60)  
result = ""  
if negative:  
    result += "-"
```

Ostatnim krokiem jest dodanie 0 na początku gdy liczba jest mniejsza od 10 w celu poprawnej grafiki. Wszystko zwracane jest za pomocą funkcji result.

Napisany kod należy odpalić w tym celu należy się posłużyć plikiem „main.py”

Do pliku importowany jest wcześniej napisany pakiet.

Najpierw należy załadować plik w tym celu służy komenda:

```
data = dataparser.load_data_from_file("Dane_Zad_3.csv")
```


Przetworzenie danych z pliku

```
items_ok, items_nok = dataparser.parse_data(data)
```

Obliczony jest czas postoju oraz opóźnienia

```
dataparser.calc_delay_and_layover_time(items_ok)
```

Zapis do pliku SQLite

```
dataparser.save_to_sqlite("result.sqlite3", items_ok)
```

Zapis do pliku CSV

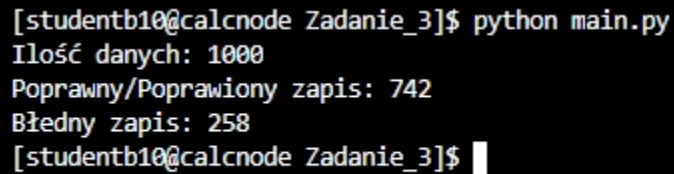
```
dataparser.save_to_csv("Not_ok.csv", items_nok)
```

W ten sposób otrzymujemy dwa nowo powstałe pliki jeden z niepoprawnymi danymi oraz drugi sqlite. Wyświetlane są również statystyki.

skrypt inicjujemy po wcześniejszej instalacji pakietu za pomocą

```
``python main.py``
```

W ten sposób otrzymujemy w konsoli:



```
[studentb10@calcnode Zadanie_3]$ python main.py
Ilość danych: 1000
Poprawny/Poprawiony zapis: 742
Błędny zapis: 258
[studentb10@calcnode Zadanie_3]$
```

Rys. 10: Statystyki po uruchomieniu main.py

Pojawiają się również dwa pliki „result.sqlite3” oraz „Not_ok”

W pliku sqlite przedstawione są poprawne dane i prezentują się one następująco.

stop_number	line_number	vehicle_number	date	time	real_time	real_departure_time	delay	layover
5	102	201	17.01.2022	06:00	06:01	06:02	01:00	01:00
10	110	753	18.01.2022	07:00	07:05	07:07	05:00	02:00
15	8	1002	19.01.2022	08:00	08:00	08:02	00:00	02:00
20	10	948	20.01.2022	09:00	09:02	09:03	02:00	01:00
25	22	1	21.01.2022	10:00	10:03	10:04	03:00	01:00
30	142	225	22.01.2022	11:00	11:01	11:03	01:00	02:00
35	145	332	23.01.2022	12:00	12:11	12:12	11:00	01:00
40	147	864	24.01.2022	13:00	13:08	13:09	08:00	01:00
45	57	321	25.01.2022	14:00	14:03	14:05	03:00	02:00
40	102	42	30.04.2021	06:21	06:31	06:34	10:00	03:00
25	22	6	06.09.2021	07:27	07:35	07:37	08:00	02:00
15	8	43	23.09.2021	02:10	02:07	02:10	-03:00	03:00
30	102	42	26.10.2021	13:54	14:00	14:01	06:00	01:00
5	102	201	10.11.2021	07:24	07:31	07:34	07:00	03:00
30	142	225	24.03.2021	19:13	19:22	19:25	09:00	03:00
10	147	864	28.09.2021	17:57	17:59	18:02	02:00	03:00
15	57	56	20.10.2021	15:31	15:38	15:41	07:00	03:00
30	102	42	20.12.2021	15:41	15:42	15:45	01:00	03:00
40	102	42	05.07.2021	08:21	08:21	08:23	00:00	02:00
25	57	321	11.08.2021	01:27	01:31	01:33	04:00	02:00
5	102	12	27.01.2021	14:35	14:40	14:42	05:00	02:00
5	102	75	16.02.2021	06:05	06:10	06:13	05:00	03:00
5	102	654	20.06.2021	16:13	16:10	16:11	-03:00	01:00
5	102	42	11.01.2021	03:46	03:45	03:46	-01:00	01:00
15	8	6	27.07.2021	15:15	15:13	15:16	-02:00	03:00
5	102	654	14.03.2021	17:27	17:33	17:34	06:00	01:00
10	147	2	23.09.2021	03:23	03:27	03:30	04:00	03:00

Rezultatem są poprawne dane

W pliku .CSV przedstawione są dane do poprawy i wyglądają następująco.

ABC 123	Numer przystanku	ABC 123	Numer linii	ABC 123	Numer pojazdu	ABC 123	Data	ABC 123	Godzina planowana	ABC 123	Godzina faktycznego przyjazdu	ABC 123	Godzina faktycznego odjazdu
1	15		8		43		17.01.2021 00:00:00		0,259027778		0,265277778	132:170	
2	50		147		234		25.08.2021 00:00:00		0,225694444		0,228472222		0,230555556
3	30		142		532		25.07.2021 00:00:00	126:255			0,339583333		0,341666667
4	15		8		6		01.08.2021 00:00:00	185:193			0,041666667		0,04375
5	52		10		948		20.06.2021 00:00:00		0,379861111		0,378472222	09:08	
6	5		102		53		12.08.2021 00:00:00		0,133333333		01.03.2019 00:00:00	155:180	
7	40		110		342		04.07.2021 00:00:00		0,538194444		0,544444444		0,545833333
8	35		243		332		13.01.2021 00:00:00		0,440972222	175:207			0,441666667
9	35		147		338		09.08.2021 00:00:00		0,754861111		0,755555556		0,756944444
10	35		145		332		27.06.2021 00:00:00		0,057638889		0,0625	188:149	
11	45		95		94		24.06.2021 00:00:00		0,597222222		0,595833333	211:207	
12	45		102		201		19.12.2021 00:00:00	181:164			0,209722222		0,210416667
13	40		147		974		21.03.2021 00:00:00		0,116666667		0,117361111		0,119444444
14	30		142		124		19.07.2021 00:00:00	138:235			0,570138889		0,571527778
15	93		102		42		03.02.2021 00:00:00		0,323611111		0,330555556		0,331944444
16	10		215		5		23.12.2021 00:00:00		0,18125		0,179166667		0,179861111
17	45		102		201		16.11.2021 00:00:00	01.04.2015 00:00:00	183:215				0,185416667
18	40		147		896		22.09.2021 00:00:00		0,749305556	127:203			0,750694444
19	30		102		42	28*11*2021		194:196			0,452777778	194:180	
20	10		147		864		25.05.2021 00:00:00	189:208			0,246527778		0,248611111
21	25		10		948		15.09.2021 00:00:00	158:158			0,127777778		0,129861111
22	5		102		716		06.08.2021 00:00:00		0,205555556		0,2125		0,214583333
23	25		22		6555		01.10.2021 00:00:00	17:36			0,7375		0,738194444
24	25		57		321		07.08.2021 00:00:00		0,007638889	160:203			0,014583333
25	30		142		532		19.11.2021 00:00:00		0,825694444	201:237			0,831944444
26	45		57		164		01.05.2021 00:00:00		0,300694444		0,3		0,301388889
27	45		57		790		08.03.2021 00:00:00		0,247916667		0,247222222		0,248611111
28	20		57		32		23.04.2021 00:00:00	128:172			0,239583333		0,241666667
29	35		196		234		05.05.2021 00:00:00		0,784722222		0,784722222		0,786111111
30	120		57		321		10.10.2021 00:00:00		0,149305556		0,15		0,151388889
31	25		22		23		04.01.2021 00:00:00	173:125			0,548611111		0,550694444
32	93		110		342		07.02.2021 00:00:00		0,79375		0,796527778		0,798611111
33	40		147		896		26.01.2021 00:00:00		0,227083333		0,229861111	152:163	

Rezultatem są nieporównane dane

Stworzony pakiet pozwala nam na sprawne oczyszczanie/korekcje danych.

1.4 Podsumowanie

Zrealizowane zadania pozwoliły na zapoznanie się z podstawowymi funkcjami analizy danych przy wykorzystaniu języka programowania python. Dodatkowo wszystkie wykonane w projekcie zadania pozwoliły na pogłębienie wiedzy z zakresu przetwarzania tekstu, odczytu i analizowaniu danych oraz tworzeniu w pełni funkcjonalnego pakietu. Analiza danych przy wykorzystaniu tego języka programowania znacznie może ułatwiać i usprawniać analizy znacznych ilości danych. Intuicyjność i szybkość wykonywania poleceń jest dwoma podstawowymi elementami przemawiającymi za jego wykorzystaniem. Jednym z najważniejszych wniosków po zrealizowaniu zadań to to, że python jest przede wszystkim językiem czytelnym i przejrzystym.