

Wykład 3. Czyszczenie i wstępne przekształcanie danych.

Michał Sochański

Plan wykładu

1. Przekształcanie typów zmiennych.
2. Czyszczenie danych.
3. Wstępne przekształcenia danych.

1. Przekształcanie typów zmiennych

Po wczytaniu zbioru często zdarza się sytuacja, że typ zmiennej nie odpowiada naszym intencjom. Może tak być np. gdy:

- 1.1. Zmienna numeryczna zostaje zakwalifikowana jako „object”.
- 1.2. Zmienna z datą zostaje zakwalifikowana jako „object”.
- 1.3. Zmienna, która jest w rzeczywistości tekstowa została zakwalifikowana jako numeryczna.
- 1.4. Chcemy potraktować zmienną typu „int” jako „float”.
- 1.5. Chcemy przekształcić zmienną typu „object” na zmienną typu „category”.

Przekształcanie typów zmiennych – tekstowe na numeryczne

1.1. Zmienna numeryczna może być zakwalifikowana jak tekstowa, jeśli pojawi się w niej choćby jedna wartość tekstowa. Można wtedy:

- Zmienić po kolei wszystkie wartości tekstowe na właściwe wartości numeryczne (jeśli je znamy).
- Zmienić wartości tekstowe na braki danych. Można to zrobić „za jednym zamachem” za pomocą następującej komendy:

```
data['wiek'] = pd.to_numeric(data['wiek'], errors='coerce')
```

Przekształcanie typów zmiennych – tekstowe na daty

1.2. Jeśli chcemy aby dana zmienna była potraktowana jako data, możemy użyć funkcji „to_datetime” z odpowiednio dobranym argumentem „format”. Np.:

```
pd.to_datetime(data['Data'], format='%m.%d.%Y')
```

W powyższym przypadku zakładamy, że zmienna „Data” ma format mm/dd/YYYY.

Przekształcanie typów zmiennych – metoda „.astype”

1.3., 1.4. Czasem chcemy potraktować zmienną wypełnioną liczbami jako tekstową, gdy liczby te odpowiadają tak naprawdę kategoriom (np. „1” to mężczyzna a „2” to kobieta)

```
data['Zmienna'] = data['Zmienna'].astype(str)
```

W podobny sposób możemy zastosować następujące komendy:

```
data['Zmienna'] = data['Zmienna'].astype(int)
```

```
data['Zmienna'] = data['Zmienna'].astype(float)
```

```
data['Zmienna'] = data['Zmienna'].astype(bool)
```

Przekształcanie typów zmiennych – tekstowe na typ „category”

1.5. Typ „category” to specjalny typ, który odpowiada zmiennym kategoryalnym. Warto ją zastosować, jeśli mamy faktycznie zmienną zawierającą kategorie, a nie tekst.

```
data['edu_cat'] = data['edu_cat'].astype('category')
```

- Można również nadać kategoriom porządek, zmieniając tym samym zmienną nominalną w porządkową

```
data['Nazwa_zmiennej'].cat.reorder_categories(['low', 'high'],  
ordered=True)
```

Przekształcanie typów zmiennych – tekstowe na typ „category”

- Najważniejsze zalety zastosowania typu „category”:
 - Łatwy dostęp do kategorii
 - Niektóre pakiety statystyczne wymagają stosowania tego typu zmiennej do poszczególnych analiz
 - Większa efektywność obliczeniowa
 - Specjalne „metody” dedykowane temu typowi zmiennej

- Przykładowe „metody” działająca na zmiennych typu „category”

`data['edu_cat'].cat.categories` #wyświetlamy wszystkie kategorie

`data['edu_cat'].cat.ordered` #sprawdzamy czy zmienna jest porządkowa

2. Czyszczenie danych

Możemy natknąć się na wiele potencjalnych problemów wczytując nowe dane. Aby przygotować zbiór do dalszych analiz, trzeba najpierw je rozwiązać, co wiąże się często z przekształcaniem wyjściowego zbioru. Najważniejsze z tych problemów to:

2.1. Braki danych.

2.2. Zduplikowane obserwacje.

2.3. Nieodpowiednie lub niespójne nazwy kolumn (zmiennych).

2.4. Nietypowe/ niepoprawne/ odstające wartości.

2.1. Braki danych

- Braki danych są bardzo powszechnym problemem i jest wiele sposobów radzenia sobie z nim. To, jaki z nich wybierzemy, zależy od konkretnej sytuacji – typu danych, typu braków danych czy celu analiz.
- Przed podjęciem decyzji odnośnie traktowania braków danych warto sprawdzić, o ile to możliwe, czy braki danych występują w jakikolwiek systematyczny sposób (czy też losowy). Jeśli tak, należy się zastanowić czy np. usunięcie danych z brakami nie wpłynie na wyniki analiz.
- Braki danych są reprezentowane a pandas przez „NaN”.
- Uwaga – braki danych są często kodowane na najróżniejsze inne sposoby, np. jako teksty „99”, „-” czy po prostu puste komórki (bądź wypełnione spacjami). Jeśli tak jest, należy przekształcić te warto się na „NaN”.

Braki danych – możliwe sposoby działania

1. Zostawiamy braki danych ale wykluczamy obserwacje z brakami z analiz dotyczących zmiennych, w których braki występują. Załóżmy na przykład, że obserwacja w danym rzędzie dotyczy Pana Jana Kowalskiego; kolumnie „wiek” mamy wartość 43, ale w kolumnie „waga” mamy brak danych. Wtedy wykluczamy tę obserwację z analiz dotyczących wagi, ale nie z analiz dotyczących wieku.

Uwaga – przy badaniu relacji dwóch zmiennych wykluczymy obserwacje, które mają braki w którejkolwiek z tych dwóch zmiennych.

2. Usuwamy wiersze, w których występują jakiekolwiek braki danych.
3. Wypełniamy braki danych określonymi wartościami, np.:
 - Średnia
 - Mediana
 - 0

Braki danych – usuwanie wierszy z brakami

Ad.2. Usuwamy wszystkie wiersze z przynajmniej jednym brakiem danych.

```
data = data.dropna()
```

Uwaga – stosując tę metodę możemy często stracić dużą część danych! Jest to więc rozwiązanie „skrajne”.

- Możemy też stworzyć wektor boolowski aby odfiltrować braki danych w konkretnej kolumnie albo w całym zbiorze. Będzie on przyjmował wartość „True”, jeśli na danej pozycji w zmiennej o nazwie „zmienna” występuje brak danych:

```
braki_zmienna = data.zmienna.isnull()
```

```
data[~braki_zmienna] #Wyświetlamy zbiór danych, ale z usuniętymi  
wierszami, w których w kolumnie o „zmienna”  
występuje brak danych.
```

Braki danych – wypełnianie braków wartościami

Ad.3.

- Wypełnianie braków danych konkretną wartością

```
data.zmienna = data.zmienna.fillna(0)
```

```
data.zmienna = data.zmienna.fillna('missing')
```

```
data[['zmienna1', 'zmienna2']] = data[['zmienna1', 'zmienna2']].fillna(0)
```

- Aby wypełnić braki danych konkretną wartością, np. średnią, należy najpierw ją wyliczyć.

```
mean_value = data.Nazwa_zmiennej.mean()
```

```
data[['zmienna1', 'zmienna2']] = data[['zmienna1',  
'zmienna2']].fillna(mean_value)
```

2.2. Zdublikowane obserwacje, błędne nazwy kolumn

Zdublikowane obserwacje możemy usunąć w następujący sposób:

```
data = data.drop_duplicates()
```

2.3. Błędne nazwy kolumn

- Nazwy kolumn można zmienić w następujący sposób:

Dla jednej kolumny:

```
data.columns.values[1] = ['Nowa_nazwa']
```

Dla wszystkich kolumn w zbiorze:

```
data.columns = ['Nowa_nazwa1', 'Nowa_nazwa2', 'Nowa_nazwa3']
```

- Nadając kolumnom nazwy warto trzymać się następujących zasad:
 - 1) Nazwy kolumn nie powinny zawierać spacji (można je zastąpić np. znakiem „_”)
 - 2) Nazwy kolumn powinny być spójne – wszystkie zacząć się od małej lub wszystkie od dużej litery.

2.4. Nietypowe/ niepoprawne/ odstające wartości.

- Jeśli zauważymy wartość, która jest niepoprawna (np. ujemny wiek), możemy ją zamienić na „właściwą” (o ile taką znamy) w *konkretnej kolumnie* np. w następujący sposób:

```
data.zmienna[data.zmienna=='błędna_wartość'] = 'prawidłowa wartość'
```

```
data.zmienna[data.zmienna== -1] = 0
```

- Jeśli chcemy zamienić *wszystkie* tego typu wartości w zbiorze, możemy np. zastosować funkcję:

```
data = data.replace('błędna_wartość', 'prawidłowa wartość')
```


2.4. Nietypowe/ niepoprawne/ odstające wartości.

- Może się zdarzyć, że musimy zastąpić jakąś wartość brakiem danych. Wtedy możemy zastosować wartość „np.NaN”, musimy się jednak upewnić, że zaimportowaliśmy pakiet NumPy.

```
data = data.replace('missing', np.NaN)
```

- Jeśli chodzi o wartości odstające, które nie są niepoprawne, można sobie z nimi radzić za pomocą konkretnych metod matematycznych (na przykład przekształcenia logarytmicznego).

3. Wstępne przekształcenia danych

3.1. Przetwarzanie zmiennych lub dodanie nowej zmiennej.

3.2. Przekształcanie typu „wide to long” oraz „long to wide”.

3.1. Przetwarzanie zmiennych lub dodanie nowej zmiennej – zmienne numeryczne

- Proste przekształcenia zmiennych numerycznych:
 - Uwaga – w tym przypadku zastępujemy istniejącą zmienną nową, przekształconą zmienną. W pierwszym przypadku po prostu dodajemy 1 do wszystkich wartości w zmiennej.

```
data.zmienna = data.zmienna +1
```

```
data.zmienna = data.zmienna*2 -5
```

```
data.zmienna = False
```

- tworzymy nową zmienną

```
data['nowa_zmienna'] = data ['zmienna'] * 2
```

```
data ['nowa_zmienna'] = data ['zmienna1'] – data['zmienna2']
```

- Uwaga – dodając nową zmienną w powyższy sposób nie możemy zastosować notacji z kropką.

3.1. Przetwarzanie zmiennych lub dodanie nowej zmiennej – zmienne tekstowe

- Zmieniamy wszystkie litery na wielkie/ małe

```
data.zmienna = data.zmienna.str.upper()
```

```
data.zmienna = data.zmienna.str.lower()
```

- Usuwamy spacje i wartości liczbowe na początku i końcu zmiennej

```
data.zmienna = data.zmienna.str.strip()
```

- Łączymy (konkatenacja) dwie zmienne tekstowe (uwaga – operator „+” działa tu inaczej niż w przypadku zmiennych numerycznych)

```
data['nowa_zmienna'] = data.zmienna1 + data.zmienna2
```

```
data['nowa_zmienna'] = data.zmienna1 + '_' + data.zmienna2
```

3.1. Przetwarzanie zmiennych lub dodanie nowej zmiennej – zmienne tekstowe

- Dodajemy zmienną, w której znajduje się wybrany znak lub podzbiór znaków należący do innej zmiennej.

```
data['nowa_zmienna'] = data.zmienna.str[1]
```

```
data['nowa_zmienna'] = data.zmienna.str[0:3]
```

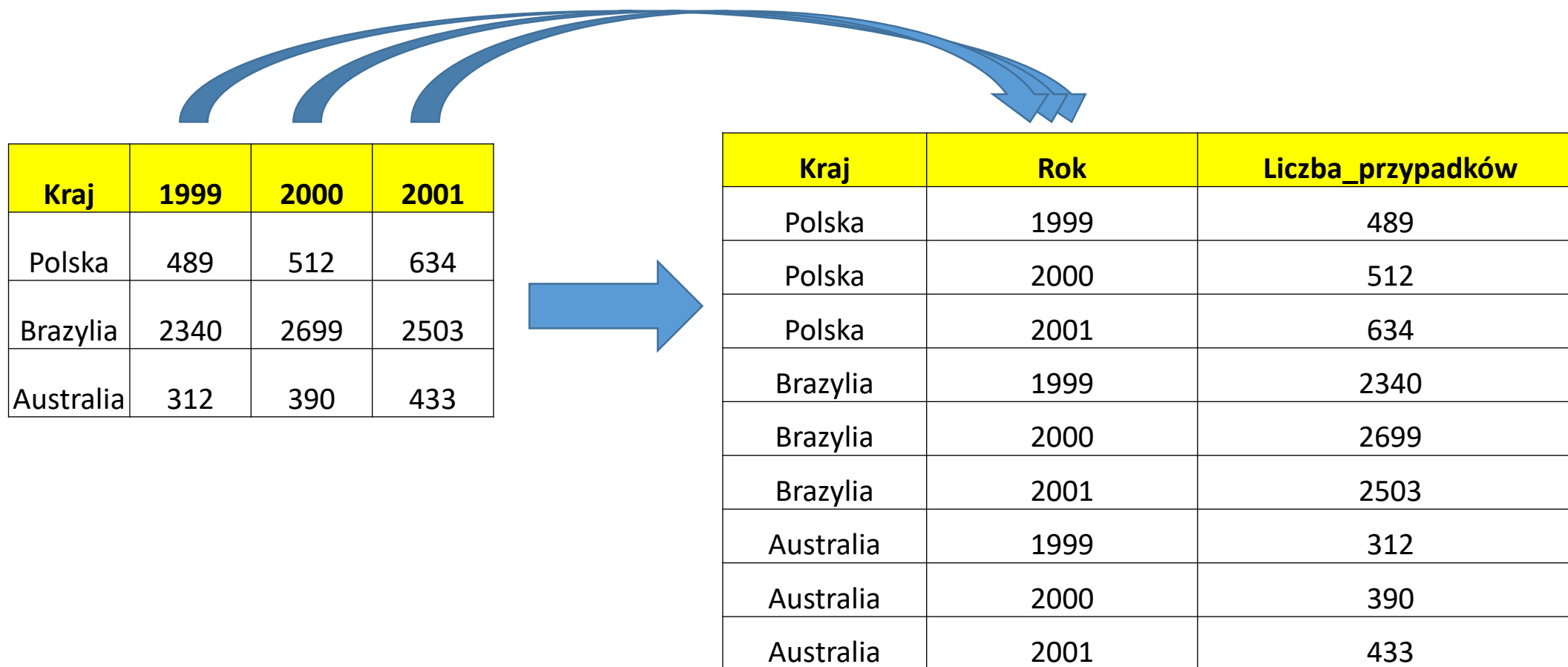
- Dodajemy zmienną, w której zastępujemy pewne znaki (występujące w wyjściowej zmiennej) innymi znakami.

```
data['nowa_zmienna'] = data.zmienna.str.replace('old', 'new')
```

3.2. Przekształcanie typu „wide to long” oraz „long to wide”.

- .
- Omówimy dwa typy przekształceń (nazwy odnoszą się do ogólnego „kształtu” zbioru – szerokiego lub długiego):
 - Wide format → Long format
 - Long format → Wide format
- Są one często (ale nie tylko) stosowane w celu przekształcania zbioru do formatu „tidy”, a więc spełniającego następujące warunki:
 - Każda kolumna odpowiada jednej „rzeczywistej” zmiennej, tzn. własności obserwacji, którym odpowiadają wiersze (nie powinno być wielu zmiennych w jednej kolumnie, ani jednej zmiennej w wielu kolumnach)
 - Każdy wiersz odpowiada jednej obserwacji

Tidy data: „wide \rightarrow long” – intuicja.




Przechodzimy od zbioru „szerokiego” do „długiego” przez „scalenie” kilku kolumn w jedną.

„wide → long” – funkcja pd.melt

```
pd.melt(data, id_vars=['Var1'],  
        value_vars=['Var2', 'Var3'],  
        var_name='Nazwa_zmiennej_scalonej',  
        value_name='Nazwa_zmiennej_z_wartościami')
```

- data – nazwa zbioru (w formacie DataFrame)
- id_vars – lista zmiennych, które będą włączone do nowego zbioru. Te zmienne pozostaną w niezmienionej postaci, tzn. nie będziemy ich łączyć z innymi kolumnami.
- value_vars – lista zmiennych, które będziemy „scalać” w jedną zmienną
- var_name – nowa nazwa „scalonej” zmiennej.
- value_name – nowa nazwa zmiennej, w której będą zawarte wartości scalanych zmiennych ze zbioru wyjściowego.

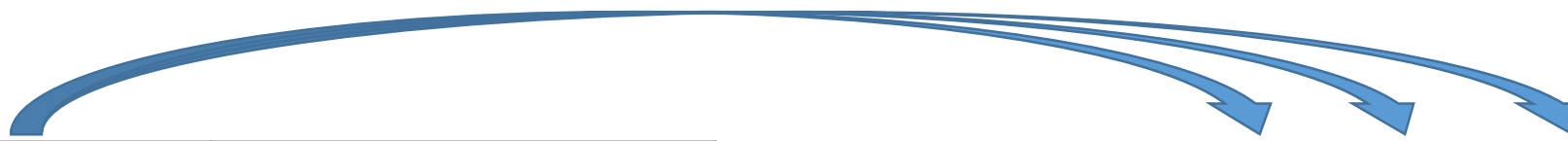


Kraj	1999	2000	2001
Polska	489	512	634
Brazylia	2340	2699	2503
Australia	312	390	433

Kraj	Rok	Liczba_przypadków
Polska	1999	489
Polska	2000	512
Polska	2001	634
Brazylia	1999	2340
Brazylia	2000	2699
Brazylia	2001	2503
Australia	1999	312
Australia	2000	390
Australia	2001	433

```
pd.melt(data, id_vars=['Kraj'],  
        value_vars=['1999', '2000', '2001'],  
        var_name='Rok',  
        value_name='Liczba_przypadków')
```

Tidy data: „long \rightarrow wide” – intuicja.



Kraj	Rok	Liczba_przypadków
Polska	1999	489
Polska	2000	512
Polska	2001	634
Brazylia	1999	2340
Brazylia	2000	2699
Brazylia	2001	2503
Australia	1999	312
Australia	2000	390
Australia	2001	433

Kraj	1999	2000	2001
Polska	489	512	634
Brazylia	2340	2699	2503
Australia	312	390	433

Przechodzimy od zbioru „długiego” do „szerokiego” przez „rozbicie” jednej ze zmiennych. Każda z jej występujących w zbiorze wartości staje się teraz nagłówkiem nowej kolumny. Dodatkowo musimy wskazać, z jakiej zmiennej mają pochodzić wartości w nowych kolumnach.

„long → wide” – funkcja pivot

```
data.pivot(index='Var1',  
            columns='Var2',  
            values=['Var3'])
```

- index – zmienna, która pozostanie niezmienną przez funkcję pivot, ale będzie użyta do etykietowania wierszy (zob. uwagi na kolejnym slajdzie)
- columns – zmienna, której wartości staną się nagłówkami kolumn w nowym zbiorze.
- values – zmienna, której wartości będą wypełniać nowe kolumny. W tym miejscu można również podać listę zmiennych. W takim przypadku kolumny z nagłówkami ze zmiennej wskazanej w argumencie „columns” będą powtórzone dla każdej zmiennej wskazanej w argumencie „values”.
- Uwaga – nazwę zbioru, inaczej niż w przypadku funkcji melt, przekazujemy przy wywołaniu funkcji – „*data.pivot*” – zamiast w jej argumencie.

Funkcja pivot - uwagi

- DataFrame, który powstaje w wyniku zastosowania funkcji „pivot” ma w etykietach wierszy (tzw. indeksach) wartości zmiennej wskazanej w argumencie „index”. Jeśli chcemy, aby ta zmienna stanowiła osobną kolumnę w zbiorze, musimy uruchomić komendę:

```
data.reset_index()
```

- Jeśli podamy więcej zmiennych w argumencie „values”, kolumnom nadane zostaną etykiety „hierarchiczne”, tzn. oprócz „zwykłych” nazw kolumn występować będą „nadrzędne” nazwy kolumn, grupujące te pierwsze. Każda grupa będzie odpowiadała jednej zmiennej wskazanej w argumencie „values”.



Kraj	Rok	Liczba_przypadków
Polska	1999	489
Polska	2000	512
Polska	2001	634
Brazylia	1999	2340
Brazylia	2000	2699
Brazylia	2001	2503
Australia	1999	312
Australia	2000	390
Australia	2001	433

Kraj	1999	2000	2001
Polska	489	512	634
Brazylia	2340	2699	2503
Australia	312	390	433

```
data.pivot(index='Kraj',
            columns='Rok',
            values=['Liczba_przypadków'])
```

Funkcja pivot - uwagi

- Aby funkcja „pivot” zadziałała musi być spełniony jeden warunek – w wyjściowym zbiorze każda para wartości z kolumn wskazanych w argumentach „index” i „columns” może występować tylko raz. W przeciwnym przypadku wartości, które mają się pojawić w zbiorze wynikowym nie będą wskazane jednoznacznie i Python wyświetli błąd.

Kraj	Rok	Liczba_przypadków
Polska	1999	489
Polska	1999	512
Polska	2000	634
Brazylia	1999	2340
Brazylia	2000	2699



Kraj	1999	2000
Polska	???	634
Brazylia	2340	2699

Polecane kursy na platformie data.camp

- Introduction to *Python*
- Importing Data in *Python* (Part 1)
- pandas Foundations
- Cleaning Data in *Python*
- Manipulating DataFrames with *pandas*
- Merging DataFrames with *pandas* #materiał ponadprogramowy –
łączenie zbiorów w pandas

Ćwiczenia

Ćwiczenie 1.

Przyjrzymy się zbiorowi „mtcars”, który jest zawarty w pliku „Wyk4_dane.xlsx”, w zakładce „mtcars”. Jest to dostępny w sieci zbiór zawierający podstawowe informacje o wybranych modelach samochodu. Zbiór pochodzi z lat 70tych, bliższa informacja o zbiorze, w tym o znaczeniu zmiennych, zawarta jest na stronie <https://stat.ethz.ch/R-manual/R-devel/library/datasets/html/mtcars.html>. W udostępnionym pliku zawarta jest okrojona i nieco zmodyfikowana wersja zbioru.

1. Wczytaj zbiór jak DataFrame korzystając z funkcji „pd.ExcelFile” i „parse”.
2. Zapoznaj się ze strukturą zbioru za pomocą metody „.info()”, albo innych opisanych na slajdach funkcji.
3. W niektórych ze zmiennych są wartości tekstowe, mimo, że zmienne powinny być numeryczne. Zamień wartości wskazujące braki danych na „NaN” (brak danych w pandas), a teksty, które powinny być liczbami na odpowiednie liczby.
4. Nazwy modeli są w etykietach wierszy. Przenieś je do osobnej kolumny za pomocą metody „.reset_index()”. Następnie zmień nazwę nowej kolumny na „model”.
5. Zastąp spacje znakiem „_” w nowej kolumnie „model” za pomocą funkcji „.str.replace()”.

Ćwiczenie 1.(cd)

6. Zmienna „am” jest zakodowana jako numeryczna, ale jest to w istocie zmienna kategoryalna („1” odpowiada ręcznej skrzyni biegów, a „0” automatycznej skrzyni biegów). Zmień ją na kategoryalną za pomocą funkcji „`astype('category')`”.
7. Przekoduj wartości w zmiennej „am” na „manual” i „automatic” (zmieniając odpowiednio wszystkie wartości na „1” i „0”).
8. W zmiennej „wt” waga jest podana w tysiącach funtów. Dodaj zmienną „wt_kg”, z wagą w kilogramach „wt_kg”. Jeden funt to ok. 0.454 kg, więc nową zmienną przypisz „starej” zmiennej przemnożonej przez odpowiednią wartość.

Ćwiczenie 1.(cd)

9. Wyświetl następujące podzbiory (dokonaj filtrowania przy użyciu odpowiednich warunków logicznych):
 - modele z automatyczną skrzynią biegów
 - modele z 8 cylindrami i pojemnością silnika („disp”) powyżej 300
 - modele, które albo mają 4 cylindry albo ważą poniżej 1 tony
10. Zapisz dane z powrotem jako plik „Excelowy” za pomocą funkcji „.to_excel(file)”.
11. Otwórz wynikowy plik „xlsx” i sprawdź czy wszystkie zmiany zostały przeprowadzone prawidłowo.

Ćwiczenie 2.

W pliku „Wyk4_dane.xlsx”, w zakładce „temp” zawarte są dane, które były omawiane na pierwszym wykładzie. Spróbuj powtórzyć czynności omówione na wykładzie, aby przejść ze struktury 1 do struktury 2 a następnie do struktury 3 (patrz kolejny slajd).

- Podpowiedź – musisz wykonać następujące czynności:
 1. Połącz kolumny d1-d5 w jedną kolumnę za pomocą funkcji „pd.melt”, pozostawiając w zbiorze wszystkie pozostałe zmienne. Nową złączoną kolumnę nazwij „day2”.
 2. Pozostaw w zbiorze tylko te wiersze, w których nie ma braków danych w zmiennej „day2”.
 3. Usuń literkę „d” ze zmiennej „day2” (w wartościach zmiennej).

Ćwiczenie 2.

4. Utwórz zmienną tekstową „Data” łączącą trzy zmienne z rokiem, miesiącem i dniem. Wartości powinny być przedzielone symbolem „.” (lub innym).
5. Utwórz zmienną „Data2”, która zawiera zmienną „Data” ale w formacie odpowiednim dla dat.
6. Pozostaw w zbiorze tylko zmienne „Data2”, „element” i „temperature”.
7. Użyj funkcji „pivot” aby przenieść wartości ze zmiennej „element” (temp_max i temp_min) do nagłówek dwóch nowych kolumn. Wartości w nowych kolumnach powinny pochodzić ze zmiennej „temperature”.
8. Na koniec posortuj dane po zmiennej „Data2”. Wynik powinien być taki, jak na kolejnym slajdzie.

(rozwiązanie tego zadania załączone jest w skrypcie z wykładu)

Dane do ćw. 2.

1.

year	month	element	d1	d2	d3	d4	d5
2014	1	temp_max	-	3	8	-	3
2014	1	temp_min	-	1	4,4	-	0
2014	2	temp_max	5	-	2	-	-
2014	2	temp_min	1	-	-4	-	-

2.

date	element	temperature
02.01.2014	temp_max	3
02.01.2014	temp_min	1
03.01.2014	temp_max	8
03.01.2014	temp_min	4,4
05.01.2014	temp_max	3
05.01.2014	temp_min	0
01.02.2014	temp_max	5
01.02.2014	temp_min	1
03.02.2014	temp_max	2
03.02.2014	temp_min	-4

3.

date	temp_max	temp_min
02.01.2014	3	1
03.01.2014	8	4,4
05.01.2014	3	0
01.02.2014	5	1
03.02.2014	2	-4

Tidy data!



Ćwiczenie 3.

W zbiorze „Titanic.csv” zawarta jest informacja o pasażerach słynnego statku, w tym o tym, ile osób z poszczególnych grup przeżyło katastrofę. Jego struktura jest specyficzna, ponieważ każdy wiersz odpowiada przecięciu kategorii z czterech pierwszych zmiennych – a więc pewnym podgrupom pasażerów – a nie poszczególnym pasażerom. W zmiennej „Freq” zawarta jest informacja o liczbie każdej z podgrup.

Założmy, że chcielibyśmy uzyskać prosty podzbiór z pasażerami trzeciej klasy, którzy przetrwali katastrofę. Chcielibyśmy jednak, aby w tym zbiorze były dwie kolumny odpowiadające odpowiednio mężczyznom i kobietom. Aby to zrobić, wykonaj następujące czynności:

1. Użyj filtru, aby pozostawić tylko pasażerów 3 klasy, którzy przeżyli katastrofę. Usuń teraz ze zbioru niepotrzebne już kolumny „Class”, „Survived” oraz „Unnamed: 0” (względnie zachowaj wszystkie kolumny poza tymi trzema). Wynikowy DataFrame zapisz w nowej zmiennej.
2. Użyj funkcji „pivot” w taki sposób, aby wartości zmiennej „Płeć” umieszczone były w nagłówkach dwóch nowych kolumn. Wartości w tych dwóch kolumnach powinny być wypełnione wartościami ze zmiennej „Freq”. W nowym zbiorze powinna również pozostać niezmienniona zmienna „Age.” Na koniec użyj komendy „.reset_index()” i sprawdź czy finalny zbiór jest taki, jak chcieliśmy.